Trabajo de Fin de Máster

Máster en Ingeniería de Automoción

Herramienta intrusiva de bus CAN

MEMORIA

Autor: Sergio González Villegas
Director: Juan Manuel Moreno Eguílaz

Convocatoria: Junio 2019

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona


ETSEIB

Resumen

La mayoría de herramientas actuales de test y validación de buses de comunicación de automoción, tales como CAN (Controller Area Network) y LIN (Local InterConnect Network), dependen del respaldo de un ordenador para funcionar.

El objetivo de este proyecto es el desarrollo de una herramienta intrusiva que sea capaz de funcionar como un nodo completo de CAN y, además, actuar como un nodo LIN. El principal requerimiento radica en que esta herramienta sea capaz de operar sin la necesidad de un ordenador, exceptuando en su programación, de manera que se gana en movilidad y autonomía.

La herramienta propuesta se ha diseñado como una shield de Arduino, de manera que es este último el que actúa como unidad de procesamiento. Se ha desarrolladoun shield en forma de circuito impreso, donde se montan los componentes necesarios para dotar a este Arduino de las comunicaciones, así como, de una interfaz de interacción de usuario.

Para el funcionamiento del dispositivo se ha elaborado una aplicación, que consiste en la lectura del bus CAN y el filtrado de los mensajes. Tras filtrar los mensajes, la aplicación reacciona a ciertos identificadores realizando un envío simultaneo de los datos recibidos por CAN a través del bus LIN implementado.

Los resultados del proyecto son muy satisfactorios, dado que la herramienta es capaz de introduciéndose en un bus CAN, leer, filtrar y modificar los mensajes circulantes, así como, en función de estos, interactuar por LIN.

Como conclusión, resaltar que tanto el desarrollo como la validación han sido satisfactorios y se ha conseguido elaborar una herramienta funcional, cumpliendo con el objetivo propuesto de conseguir un dispositivo de tamaño compacto que no requiere de un ordenador.


ETSEIB

Sumario

1. Glosario

CAN: Controller Area Network

CAN FD: Controller Area Network Flexible Data-Range

GND: Ground

HMU: Human-Machine Interface

LED: Light-Emmiting Diode

LIN: Local Interconnect Network

MCU: Micro Controller Unit

OLED: Organic Light-Emmiting Diode

PCIe: Peripheral Component Interconnect Express

RF: Radio Frequency

SMD: Surface Mounted Device

2. Prefacio

2.1. Motivación

La electrónica está tomando un fuerte protagonismo cada vez mayor en el mundo de la automoción. Las nuevas tendencias dentro del sector y las nuevas demandas por parte de los consumidores están haciendo evolucionar el automóvil desde un vehículo puramente mecánico a un vehículo donde la electrónica está presente en todas y cada una de las partes que lo componen.

Esta evolución crea la necesidad de establecer comunicaciones entre las diferentes electrónicas que podemos ubicar dentro del vehículo mediante protocolos que se adapten a las necesidades del sector.

Este proyecto encuentra su motivación en explorar las posibilidades que tendría un usuario de poder comunicarse con el propio vehículo a partir de los propios buses que en éste habitan.

2.2. Antecedentes

En el mercado actual, debido a la fuerte presencia de buses de comunicación en el vehículo, encontramos diversas soluciones de comunicación ofrecidas por un amplio conjunto de empresas y proveedores. Entre ellas, el líder con más cota de mercado es Vector Informatik [1], con una amplia gama de productos tanto de hardware como de software para la comunicación con el vehículo.

La gran mayoría de empresas del sector utilizan los paquetes de software de Vector Informatik. Entre ellos, los más conocidos y utilizados son CANoe, CANape y CANalyzer. Estos paquetes de software sirven para analizar, diagnosticar, comunicar, testear y simular con los diferentes buses de comunicación de un vehículo.

Para la utilización del software de Vector Informatik es necesario disponer, además de las licencias oportunas, de su hardware. Su hardware, comúnmente denominado CANcase, proporciona la interfaz física para poder conectar el bus de comunicación pertinente del vehículo con un ordenador. Este hardware es por lo general bastante costoso y en función de su complejidad, incluye más o menos buses por dispositivo.

La VN1530 (Fig. 1), de Vector Informatik, es una tarjeta PCIe que permite instalarse de manera fija en un PC, dotándolo de 4 canales de CAN FD y hasta dos canales más de CAN o LIN. Existen otras soluciones de hardware que funciona mediante USB [1].

Fig. 1 – VN1530 de Vector Informatik Fuente: [1].

Otra de las grandes opciones ofrecidas por *Vector Informatik* son las interfaces por USB. Podemos encontrar una gran variedad, desde solo dos canales CAN o LIN, hasta varios de ellos combinados en el mismo hardware.

La VN7640 (Fig. 2) es de los dispositivos más completos que ofrece Vector Informatik. Cuenta con CAN FD, LIN, FlexRay y Ethernet. Esta herramienta es de las más costosas, ya que ofrece varios puertos de FlexRay y CAN FD en función de cómo se configure, además de interface Ethernet y LIN. A diferencia de la anterior, ésta tiene formato portable e interactúa con el PC por comunicación USB [2].

Fig. 2 - VN7640 de Vector Informatik. Fuente [2]

Todas las herramientas mencionadas anteriormente necesitan, en general, de un PC para procesar los datos que envían e interactuar con ellos, además de softwares específicos. La idea de este proyecto es el desarrollo de un dispositivo que, sin necesidad de PC, sea capaz de hacer este procesamiento de los datos, obteniendo una solución compacta, autónoma y sobre todo muy económica.

Explorando el mercado, no se han encontrado herramientas de este estilo, pues como las que se han mencionado anteriormente serían configurables para muchos usos, este tipo de herramientas objeto de este proyecto han de ser más personalizadas y específicas para la función que se quiere realizar.

3. Introducción

3.1. Objetivos del proyecto

Este proyecto tiene como objetivo principal el diseño y desarrollo de un dispositivo electrónico capaz de interactuar con los buses de comunicación de un vehículo, con la particularidad de no necesitar la acción externa de un PC, sino de ser un dispositivo totalmente portátil y autónomo.

A su vez, otro objetivo del proyecto es que el dispositivo ofrezca una interacción con el usuario mediante una interfaz HMU *(Human-Machine Interface).* Para ello, se buscarán soluciones que se puedan integrar en el dispositivo acorde a los objetivos y requisitos establecidos, como por ejemplo un tamaño reducido y portabilidad.

En cuanto a funcionalidades en comunicación, el dispositivo ha de contar con dos nodos CAN completamente independientes, de manera que, mediante selectores, se puedan seleccionar dos modos de funcionamiento distintos, uno en donde interviene el microcontrolador y otro en que no.

Además de dos nodos CAN, ha de ofrecer un nodo LIN, capaz de enviar y recibir mensajes independientemente de los nodos CAN.

Para ello, se estudiarán y valorarán diversas posibilidades, hasta llegar a la más óptima en torno a los objetivos establecidos.

3.2. Alcance del proyecto

El alcance del proyecto es el diseño y desarrollo de toda la interfaz que permita la comunicación con los buses del vehículo y la interacción con el usuario. El diseño completo del hardware (PCB) de la parte de CPU y su procesamiento no es alcance de este proyecto. Para esta parte se estudiarán qué posibilidades podemos encontrar en el mercado, para adecuarlo lo mejor posible a nuestros requisitos de diseño. Sí es objetivo de este proyecto la interacción de la parte de CPU y procesamiento con la parte de comunicaciones que se diseñe.

3.3. Modos de funcionamiento

Un punto clave de este dispositivo son sus modos de funcionamiento en cuanto a comunicaciones CAN se refiere. Como se ha mencionado anteriormente, cuenta con dos. El primer modo de funcionamiento actúa como un *sniffer* del bus, es decir, monitoriza los mensajes que se están enviando por éste. En cambio, el segundo modifica los mensajes existentes por el bus o añade nuevos.

- **Modo 1:** Las tramas que entran por el puerto de entrada son transferidas al puerto de salida y a la unidad de procesamiento.
- **Modo 2:** Las tramas que entran por el puerto de entrada son procesadas por el microcontrolador, pudiendo eliminar, añadir o modificar las mismas.

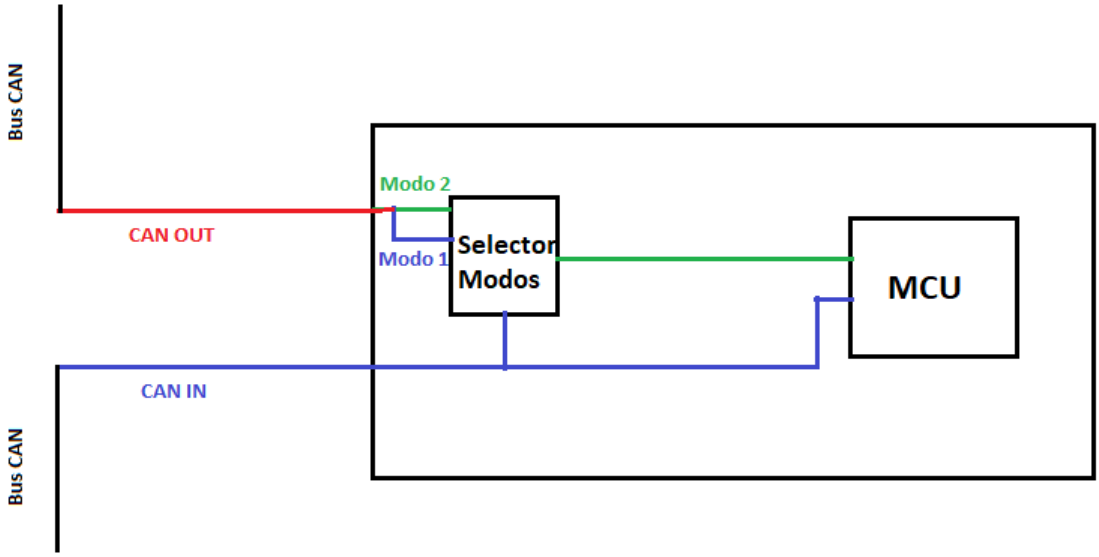


Fig. 3 - Esquema de modos Fuente: propia.

En la Fig. 3 podemos ver un esquema de cómo es el funcionamiento. El dispositivo se añade al bus como un nodo cualquiera. El bus entra por CAN IN y dependiendo del modo, lo que tenemos en la salida, CAN OUT, es la trama modificada por el MCU (Modo 2) o la trama simplemente volcada de CAN IN (Modo 1).

4. Hardware

4.1. Unidad de procesamiento

Para la unidad de procesamiento, tal y como se ha mencionado con anterioridad, se buscarán diferentes opciones en el mercado para ver cuál se adapta mejor a los requisitos del proyecto.

4.1.1. Requisitos mínimos

Como unidad de procesamiento, se necesita una tarjeta de desarrollo basada en microcontroladores, que nos ofrezca la suficiente velocidad para trabajar con el bus CAN y LIN, además de conectividad con PC y puertos libres para poder implementar diversas funcionalidades.

Dado que va a ser la base del diseño, también buscamos optimizar el tamaño y la distribución de pines, por lo que también se tendrá en cuenta. Por último y más importante, al ser un dispositivo que deberá funcionar por si solo en el entorno de automoción, debemos garantizar su funcionamiento a las tensiones de batería, es decir 12 V.

Desglosando los requisitos, podríamos resumirlos así:

- Alimentación 12 V.
- CPU suficientemente potente para la comunicación en tiempo real.
- Memoria RAM suficiente.
- Facilidad para comunicación CAN y LIN.
- Puertos extra para otras funciones.
- Dimensiones.

4.1.2. Opciones en el mercado y elección

Con los requisitos mínimos claros, se realiza una búsqueda de hardware que nos permita implementar el proyecto.

Para ello, se enfoca la búsqueda en tarjetas de desarrollo basadas en microcontroladores, ya que se considera que es un buen dispositivo para nuestro propósito, al ofrecer en una estructura compacta, puertos de entrada y salida, convertidores, comunicaciones y un largo etcétera.

Lo más común dentro de este ámbito son las tarjetas de Arduino [3] y Raspberry [4]. Estos dos fabricantes ofrecen varios modelos de tarjetas. Debido a nuestros requisitos, especialmente en comunicaciones, se tendrá que recurrir a por los modelos de gama más alta.

Dentro de Arduino tenemos el Arduino Due. Éste cumple las siguientes especificaciones: [3]

Microcontrolador	AT91SAM3X8E
Voltaje de alimentación	7-12 V
Memoria	512 kB
Pines I/O	54
Dimensiones	101 x 53
CAN	2 x CAN Controller
LIN	LIN Controller

Tabla 1 - Características Arduino. Fuente [3]

Como podemos apreciar, este modelo de Arduino cumple con todos los requisitos necesarios del proyecto. Si nos centramos en Raspberry [4], el modelo Pi 3 B+ es también válido para el proyecto. Sus características son las siguientes:

Microcontrolador	**Cortex-A53 (ARMv8) 6**4-bit SoC @ 1.4GHz
Voltaje de alimentación	5 V
Memoria	Micro SD
Pines I/O	40
Dimensiones	85 x 56
CAN	No incluido
LIN	No incluido

Tabla 2 - Características Raspberry. Fuente [4]

Como podemos apreciar, Raspberry Pi 3 B+ es mucho más potente que el modelo de Arduino. Sin embargo, no nos ofrece las mismas facilidades para poder implementar la comunicación CAN y LIN, que son objetivos básicos de este proyecto. Con Raspberry necesitaríamos añadir los controladores que, en el caso de Arduino, lleva ya integrados en su hardware.

Finalmente, tras analizar los dos modelos, se escoge Arduino Due por los siguientes motivos:

- **Alimentación:** Los 12 V de alimentación es requisito fundamental.
- **Microprocesador:** La CPU tiene potencia suficiente para nuestro propósito
- **Comunicaciones:** Incorpora controladores de CAN y LIN, nos facilita mucho el trabajo de comunicación, reduciendo el hardware externo necesario.

Por lo tanto, se procede a diseñar la capa de hardware entorno a la tarjeta Arduino Due (Fig. 4).

Fig. 4 - Arduino Due. Fuente [3].

4.2. Shield. Concepto y funcionalidades

4.2.1. Concepto

La capa de hardware para la tarjeta desarrollada en este proyecto se diseña a medida en forma de shield [5]. Se ha considerado que es la mejor opción para poder optimizar tamaño y aprovechar la forma base de nuestra tarjeta. Al desarrollar la capa en forma de shield, se debe prestar especial atención a la disposición de pines del Arduino Due, por el hecho de que condicionará de forma muy importante el diseño y la ubicación de componentes.

Una shield, en nuestro caso, es una PCB diseñada a medida, de manera que pueda ser insertada encima de los pines de entrada/salida (I/O) de nuestro Arduino Due, para que queden unidas como un nivel más de superficie por encima, añadiendo las funciones que se quieran implementar.

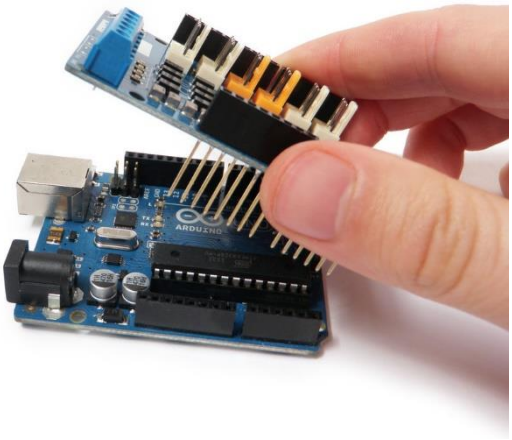


Fig. 5 - Concepto de Shield. Fuente [5].

Como se puede apreciar en la Fig. 5, la shield se inserta en los pines de entrada/salida del Arduino, dotándolo de nuevos componentes hardware y nuevas funcionalidades.

4.2.2. Funcionalidades

La shield propuesta debe dotar al Arduino de ciertas funcionalidades nuevas, tanto relacionadas con las comunicaciones como con la interacción con el usuario.

Potencia

Según las especificaciones de Arduino Due [3], éste puede ser alimentado a través de su pin Vin desde 7 V a 12 V. Como la finalidad de este dispositivo es poder funcionar de manera autónoma dentro de un vehículo, tenemos que tener en cuenta que la tensión disponible en este caso serán los 12 V de la batería.

La batería del vehículo no es una fuente de tensión muy constante, por lo que tienen ligeras fluctuaciones durante su funcionamiento y arranque, pudiendo registrar picos de hasta 16 V.

Para ello, además del regulador interno que presenta Arduino de hasta 12 V, se instala uno en la shield diseñada, de manera que pueda absorber estos picos que la batería puede causar y subministrar una tensión más constante de 12 V.

Por todo ello se propone instalar un regulador de tensión de baja caída (LDO), así como, alguna medida de protección contra inversión de polaridad para proteger el sistema. Cabe recordar que, de los 12 V de la batería, además de Arduino, se alimenta la electrónica relativa al LIN bus.

Las demás tensiones necesarias para hacer funcionar el sistema, 5 V y 3,3 V, son suministradas por el propio Arduino Due a través de los reguladores internos de los que dispone. Por eso, es de vital importancia que éste reciba una tensión regulada de 12 V para su correcto funcionamiento y, por ende, el de toda la electrónica.

Comunicaciones

El Arduino Due, como se ha mencionado anteriormente, cuenta con controladores de CAN y LIN, pero para establecer una comunicación correcta con el bus, además de controladores, necesitamos los correspondientes transceptores.

Los controladores, por así decirlo, son los encargados de transformar las tramas de comunicación proporcionadas por el microcontrolador al formato del protocolo correspondiente, en este caso CAN y LIN. El transceptor, en cambio, es el elemento encargado de proporcionar a estas tramas los niveles de tensión adecuados a la capa física establecida por el estándar. Por así decirlo, sin el transceptor, el Arduino nos proporcionaría

tramas CAN y LIN correctas, pero no con los niveles de tensión adecuados, sino que las proporcionaría a 3,3 V y 0 V, que es su tensión de funcionamiento.

En el caso del bus CAN, los niveles de tensión están entre 3,5 V y 1,5 V para CAN High y CAN Low. Por lo tanto, el transceptor, se encarga de adaptarlos.

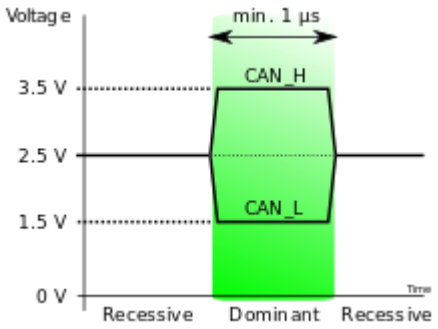


Fig. 6 - Niveles tensión CAN. Fuente [6]

En el caso del bus LIN, sus niveles de tensión son 0 V y 12 V, por lo que el transceptor de encargará de elevar la tensión desde 3,3 V a 12 V cuando sea necesario.

Además de los transceptores, será necesario proporcionar un o varios conectores para poder conectar nuestros nodos con otros nodos exteriores.

Las funcionalidades van estrechamente relacionadas con los modos de funcionamiento vistos anteriormente.

En el modo 1, el funcionamiento es de *sniffer*. Podemos leer tramas de comunicaciones, monitorizar el bus y realizar acciones por LIN en función de las tramas leídas.

En el modo 2, además de las mencionadas anteriormente, tenemos la opción de modificar las tramas entrantes. Es decir, podemos falsear datos, añadir nuevos mensajes o eliminar existentes.

El modo 1 necesita únicamente de la utilización de un puerto CAN por parte del dispositivo, mientras que el modo 2 necesita de ambos funcionando, uno para leer y otro para escribir. En el modo de funcionamiento 1, el volcado de información desde CAN IN a CAN OUT se hará mediante simples interruptores en el selector de modos.

En la tabla siguiente podemos ver resumidas las funciones que tiene cada modo y cuántos recursos utiliza, tanto a nivel de hardware como de procesamiento.

	Modo 1	Modo 2
Monitorización	✓	✓
Modificación de mensajes	×	✓
Eliminación de mensajes	×	✓
Adición de mensajes	×	✓
Interacción con LIN	✓	✓
Demanda MCU	Baja	Alta

Tabla 3 - Modos de funcionamiento. Fuente propia.

Interacción con el usuario

A parte de las comunicaciones, otro de los objetivos del proyecto es proporcionar una interacción con el usuario, por lo que se añade una pantalla o display tipo OLED, capaz de mostrar información y una serie de pulsadores para poder interactuar con el dispositivo. De esta manera, con el display se obtiene información de parte del dispositivo y con los pulsadores, se puede controlar y configurar.

4.3. Shield. Componentes clave

Para el diseño de la shield son necesarios una gran variedad de componentes electrónicos. Algunos de estos elementos son claves para el diseño, por lo que se describen a continuación para destacar brevemente sus especificaciones y motivo de elección.

Como se ha mencionado anteriormente, el Arduino Due cuenta con controladores CAN y LIN, pero, sin embargo, no cuenta con transceptores. Por lo tanto, estos elementos son clave en el diseño propuesto. Los transceptores completan, junto con el Arduino Due, la parte de comunicaciones del dispositivo desarrollado en este proyecto.

4.3.1. Transceptor CAN

Teniendo en cuenta que Arduino ofrece dos nodos CAN independientes y ya que para las funcionalidades descritas necesitaremos utilizar los dos nodos CAN, es interesante buscar un

circuito integrado, que en un mismo encapsulado, nos ofrezca dos transceptores, de manera que se pueda ahorrar espacio y coste.

Realizando una búsqueda con los filtros adecuados, encontramos un circuito integrado que cumple las características anteriormente mencionadas, en concreto el modelo MCP25612FD de Microchip [7].

El circuito integrado MCP25612FD es descrito por Microchip como un *Dual CAN Flexible Data-Rate Transceiver.* Esto quiere decir que es un integrado que soporta dos nodos CAN y, además, soporta CAN FD. En nuestro diseño no se va a utilizar CAN FD, pero para un futuro se podría estudiar cómo adaptarlo. A continuación, se detallan algunas características relevantes de este circuito integrado [7].

- Soporta CAN 2.0 y CAN FD.
- Dos alimentaciones independientes por transceptor.
- Compatible con microcontroladores de 5 V.
- Encapsulado de 14 pines en formato SOIC.
- Protección contra transitorios en automoción.
- Protecciones térmicas y de cortocircuito.
- Temperatura de funcionamiento -40 ºC +125 ºC.
- Cumple normativa específica para implementación en automoción.

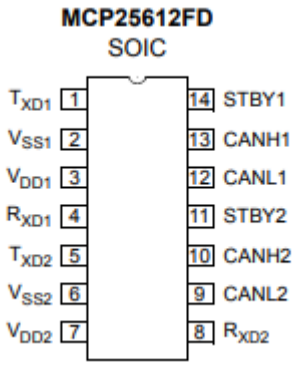


Fig. 7 - Encapsulado MCP25612FD. Fuente [7]

La hoja de características o *datasheet* del fabricante nos facilita un esquemático de aplicación típica, tal y como se muestra en la Fig. 8.

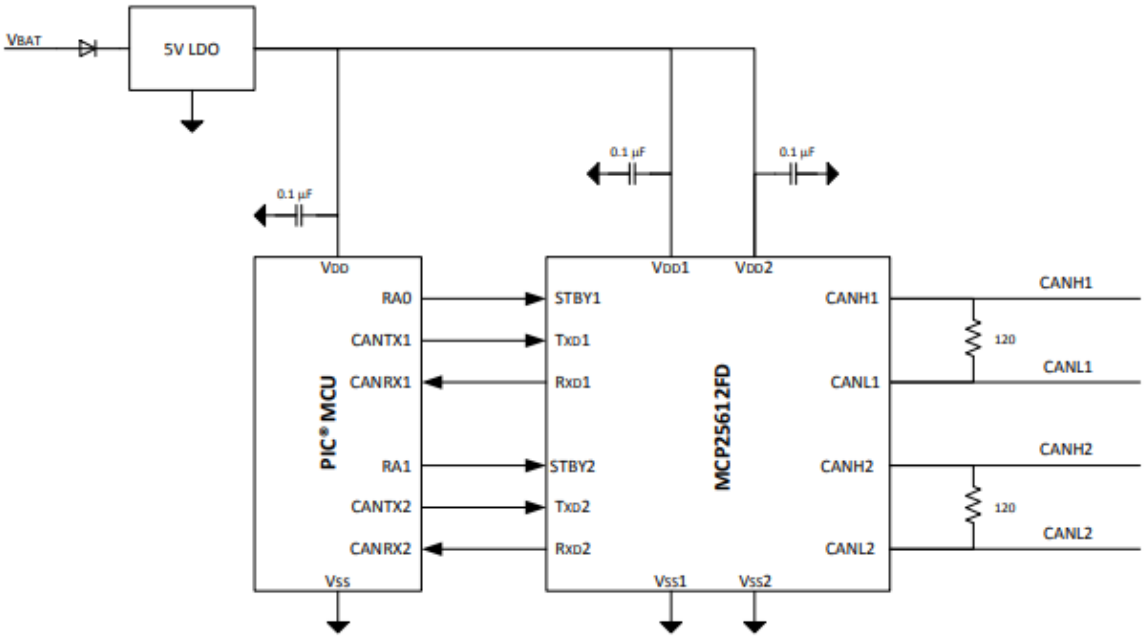


Fig. 8 - Aplicación típica MCP25612FD. Fuente [7]

En la Fig. 8 se puede observar cómo para una aplicación de dos nodos CAN, necesitamos básicamente alimentación a 5 V, la comunicación CAN y un par de pines extras en el MCU. Todos estos requisitos los cumplimos, por lo que este transceptor es un buen candidato para nuestro diseño.

4.3.2. Transceptor LIN

En este caso únicamente necesitamos un transceptor LIN, al disponer únicamente de un nodo. El transceptor que buscamos ha de poder funcionar con las tensiones que ofrece Arduino en alimentación y comunicación. Para la comunicación deberá ser compatible con 3,3 V y alimentación entre 3,3 V y 12 V.

El mismo fabricante que el transceptor de CAN, Microchip, ofrece interesantes soluciones ampliamente utilizadas en la industria de automoción. En nuestro caso, nos hemos decantado por el modelo MCP2004 de Microchip [8]. A continuación, se citan las características más relevantes para su elección.

- Es compatible con LIN 1.3, 2.0 y 2.1.
- Conforme a las normativas sobre el bus LIN.
- Gran inmunidad a EMI y ESD, así como interferencias RF.
- Alimentación entre 6 V y 27 V.
- Encapsulado de 8 pines en formato SOIC.
- Protecciones térmicas y de cortocircuito.
- Temperatura de funcionamiento -40 ºC +125 ºC.
- Cumple normativa específica para implementación en automoción.

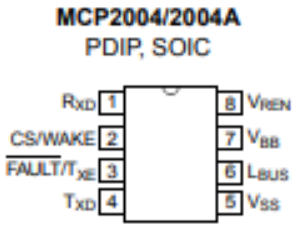


Fig. 9 - Encapsulado MCP2004. Fuente [8]

De forma análoga que el modelo anterior, el fabricante nos proporciona un esquemático de aplicación típica, tal y como puede apreciarse en la Fig. 10.

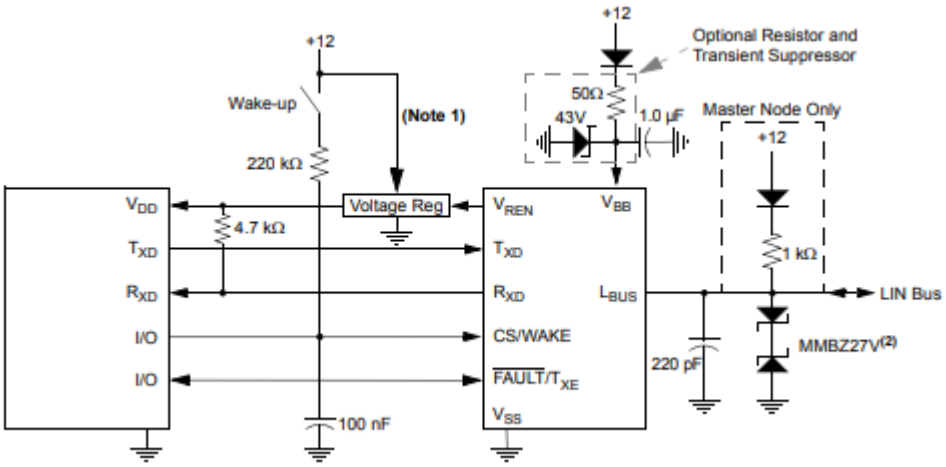


Fig. 10 - Aplicación típica MCP2004. Fuente [8]

Tal como se puede apreciar en la Fig. 10, se necesitan de los pines de comunicación provenientes de nuestro MCU, de dos pines de entrada/salida I/O adicionales y de la alimentación. Por lo tanto, podemos implementarlo correctamente en nuestro diseño.

4.3.3. Display OLED

Los transceptores mencionados anteriormente son elementos clave para la parte de comunicaciones de nuestro dispositivo. En cambio, para la parte de interacción con el usuario, el componente clave es una pantalla o display OLED para establecer la comunicación dispositivo-usuario y una serie de pulsadores para la comunicación usuario-dispositivo.

Con la máxima de reducir tamaño, se busca un único dispositivo que integre el display OLED y los pulsadores en el mismo formato, de tal manera que se pueda optimizar espacio de diseño.

En este caso, ADAFRUIT INDUSTRIES [9] tiene en su catálogo un dispositivo formado por una pantalla OLED y tres pulsadores específicamente diseñado para su placa de desarrollo FeatherWing, que es una placa de desarrollo basada en microcontroladores de la marca ADAFRUIT. Por nuestra parte, igualmente podremos adaptar este display a nuestro diseño, de manera que podamos aprovechar sus características y su reducido tamaño.

El modulo funciona a 3,3 V y se comunica con el microcontrolador mediante I2C, por lo que es un candidato perfecto para nuestro dispositivo.

El módulo de pantalla OLED, en concreto, se muestra en la Fig. 11.

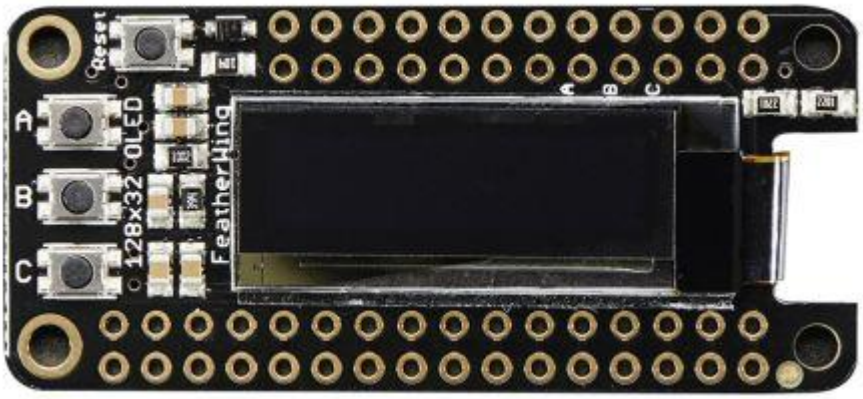


Fig. 11 - Módulo OLED ADAFRUIT. Fuente [9].

Como se puede observar en la Fig. 11, cuenta con un display OLED en el centro de 128 x 32 pixeles, además de tres pulsadores denominados A, B, C, que se pueden utilizar para asignarlos a cualquier pin I/O de nuestro Arduino Due e interactuar con él.

El display que monta el módulo está controlado por el circuito integrado SSD1306. Este chip, en concreto, se describe como un driver con controlador para matrices LED. Este dato es importante tenerlo en cuenta a la hora de realizar el software para un futuro, ya que deberá controlarse como un SSD1306.

Resumiendo, las características que nos aporta este módulo a nuestro diseño quedarían de la forma:

- Funcionamiento a 3,3 V.
- Comunicación por I2C.
- 3 pulsadores + reset.
- OLED 128x64 pixeles.

Como se puede apreciar, el funcionamiento a 3,3 V es compatible con nuestro diseño. En el caso de la comunicación I2C tenemos los puertos disponibles en el Arduino Due. El tamaño del display es suficiente para mostrar varias líneas de texto de manera clara y legible, además de los pulsadores que nos proporcionan interacción.

4.4. Shield. Diseño PCB.

El diseño de la PCB es un punto clave en este proyecto. Por lo tanto, hay que prestarle especial atención porque es la capa física sobre la cual funcionará nuestro dispositivo.

Para el diseño de la PCB se emplea la suite de diseño EAGLE [10], que nos proporciona un entorno especifico tanto para el diseño del esquemático como del layout de la PCB. Además, cuenta con una licencia de estudiante mediante la cual tendremos acceso completo a todas las funcionalidades que nos ofrece el software.

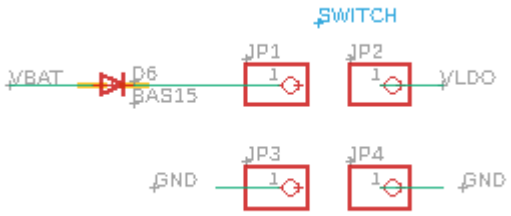
4.4.1. Esquemático

El esquemático del diseño electrónico del hardware lo podemos encontrar en el anexo adjunto. En este apartado se van a describir los bloques que lo forman y los componentes claves, así como los pines más importantes a tener en cuenta de Arduino.Due.

Para empezar, el esquemático está dividido en tres grandes bloques:

- Bloque de potencia
- Bloque de CAN
- Bloque de LIN

El bloque de potencia es el bloque encargado de proporcionar una correcta alimentación al hardware. Está formado por una protección contra inversión de polaridad y por un regulador de tensión (LDO) para evitar una tensión excesiva.



Figa. 12 - Bloque potencia 1. Fuente propia

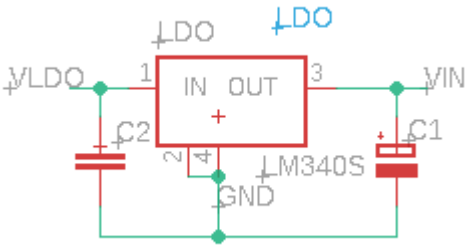


Fig. 13 - Bloque potencia 2. Fuente propia.

En las Figs. 12 y 13, se pueden observar los dos elementos que componen el bloque de potencia. El primero de todos, como se ha mencionado anteriormente, es el diodo de protección contra inversión de polaridad. Este diodo está situado junto a la entrada de alimentación y justo antes del interruptor de encendido.

La Fig. 13 corresponde con el regulador de tensión. Este regulador está situado después del interruptor y es el encargado de proporcionar una tensión de 12 V. Está junto a sus condensadores de filtrado.

Tras este regulador de tensión, conectamos directamente con los pines de alimentación de Arduino, así como GND.

El bloque de CAN es el bloque encargado del transceptor CAN y de todos los componentes necesarios para su correcto funcionamiento.

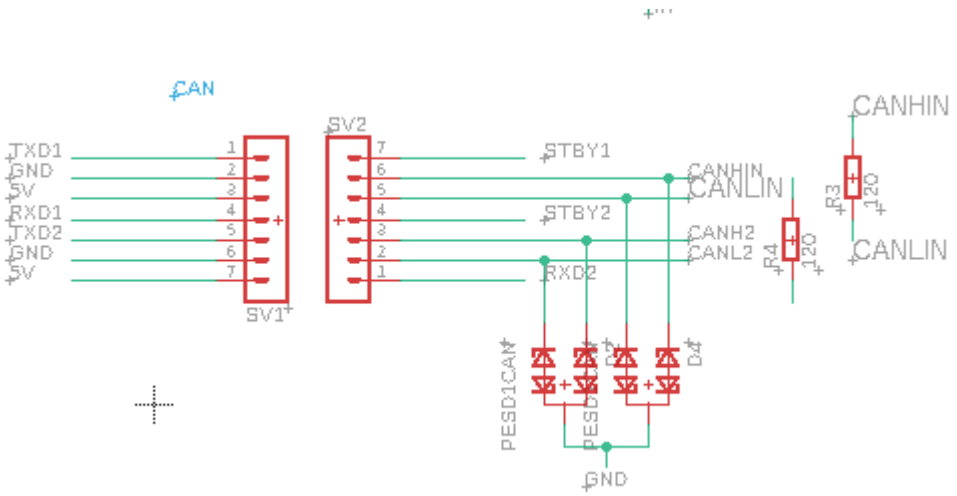


Fig. 14 - Bloque CAN. Fuente propia.

Como se puede apreciar en la Fig. 14, el bloque CAN está compuesto por el zócalo donde irá insertado el transceptor y sus componente auxiliares.

Se añaden dos resistencias de 120 Ω que actuarán como terminación de línea, entre CAN High y CAN Low de ambos canales. Además, se puede apreciar la inclusión de unos diodos específicos en cada una de las líneas. La función de éstos no es más que mejorar la estabilidad de la comunicación y proteger contra ESD.

Por último, el bloque LIN, al igual que el bloque anterior, está formado por el zócalo para el transceptor y por sus elementos auxiliares para funcionar.

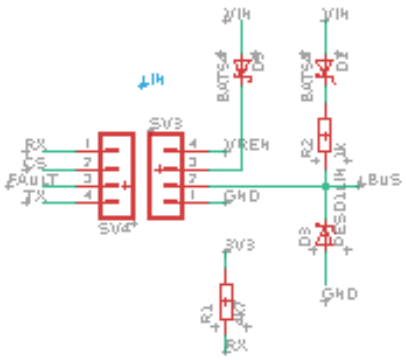


Fig. 15 - Bloque LIN. Fuente propia.

En la Fig. 15, se puede observar el bloque anteriormente descrito. A diferencia del anterior, éste está conectado a una tensión de 12 V, por lo que se han añadido dos diodos para mejorar la estabilidad de ésta.

Además, como en el caso anterior, dispone también de un diodo específico para la protección contra ESD en el caso de comunicación LIN.

El diseño está formado por más bloques, como el de la pantalla OLED o los conectores de entrada. Por la simplicidad de estos no se entra en detalle en este documento, pero se pueden encontrar los esquemáticos completos en los anexos.

Es muy interesante también resumir en una tabla la conexión de cada elemento con el pin correspondiente de Arduino, de cara a cuando se escriba el código, poder saber a qué pin debemos referirnos. A continuación, se muestra la tabla mencionada.

Conexiones esquemático	Pines Arduino
RST (OLED)	Reset
SCL (OLED)	SCL (21)
SDA (OLED)	SDA (20)
A (OLED)	10
B (OLED)	9
C (OLED)	8
RX (LIN)	Tx1 (18)
TX (LIN)	Rx1 (19)
CS (LIN)	7
FAULT (LIN)	6
RXD1 (CAN)	CANRX
TXD1 (CAN)	CANTX
RXD2 (CAN)	DAC0
TXD2 (CAN)	53

Tabla 4 - Correspondencia Pines. Fuente propia.

4.4.2. Layout. Restricciones físicas del diseño

Siguiendo la filosofía de que se está diseñando una shield específica para un Arduino Due, tendremos unas restricciones físicas en cuanto a tamaño y posicionamiento de algunos componentes.

En cuanto a tamaño, la shield propuesta no debe superar en exceso el tamaño de la tarjeta Arduino, por lo que la primera restricción es en cuanto a las dimensiones de ésta.

Según las especificaciones de Arduino, éste cuenta con unas dimensiones de 101 mm x 53 mm, por lo que nuestro tamaño debe ser similar.

La segunda y más importante condición de diseño es la ubicación de los pines que conectan nuestra capa de hardware con la tarjeta Arduino. Eagle cuenta con una librería específica para la creación de shields para Arduino, por lo que el mismo software nos proporciona ya el layout de la distribución de estos pines.

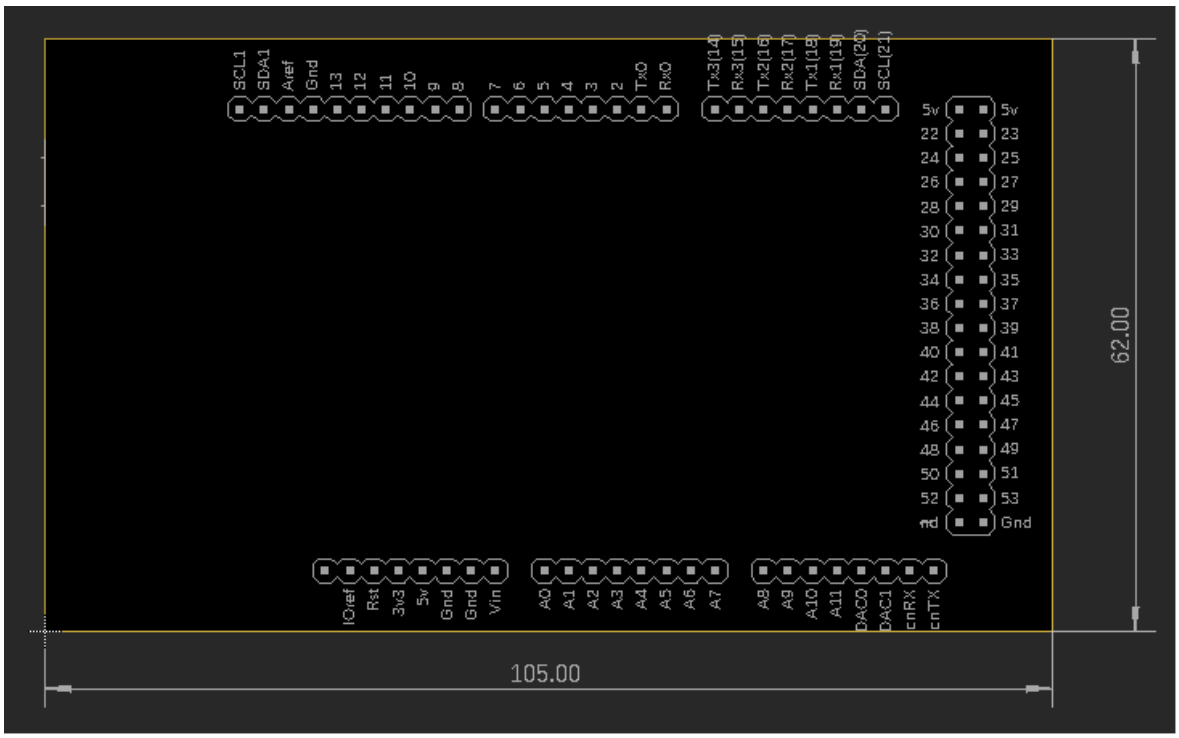


Fig. 16 – Dimensiones de la shield y posicionamiento de los pines. Fuente propia.

Tras situar la distribución de los pines y ajustar el tamaño de la PCB, nuestro diseño queda bloqueado con unas dimensiones de 105 mm x 62 mm, tal y como vemos en la Fig. 16.

A partir de aquí se proceder a ubicar los componentes por orden de importancia.

En primer lugar, se procede a ubicar los componentes mecánicos, que son los que restringen el ruteado y toda la colocación de los componentes pasivos y SMDs.

Varios componentes, como integrados de tamaño reducido, son montados mediante unos zócalos para facilitar el montaje y abaratar costes de PCB, al poder usar así, un tamaño de pista mayor.

Estos componentes son básicamente dos, los dos transceptores de CAN y LIN, respectivamente.

Mediante tiras de pines se forma un zócalo donde se insertan estos SMDs a través de una PCB adaptadora. En la Fig. 17, se muestra ilustrado.

Fig. 17 - Zócalo circuitos integrados. Fuente propia

Los componentes mecánicos que se colocan son los siguientes:

- Pin headers CAN
- Pin headers LIN
- Pin headers pantalla OLED
- DB9
- Interruptor selector modos
- Interruptor ON/OFF

Tras ubicarlos en la PCB, se tiene la siguiente distribución:

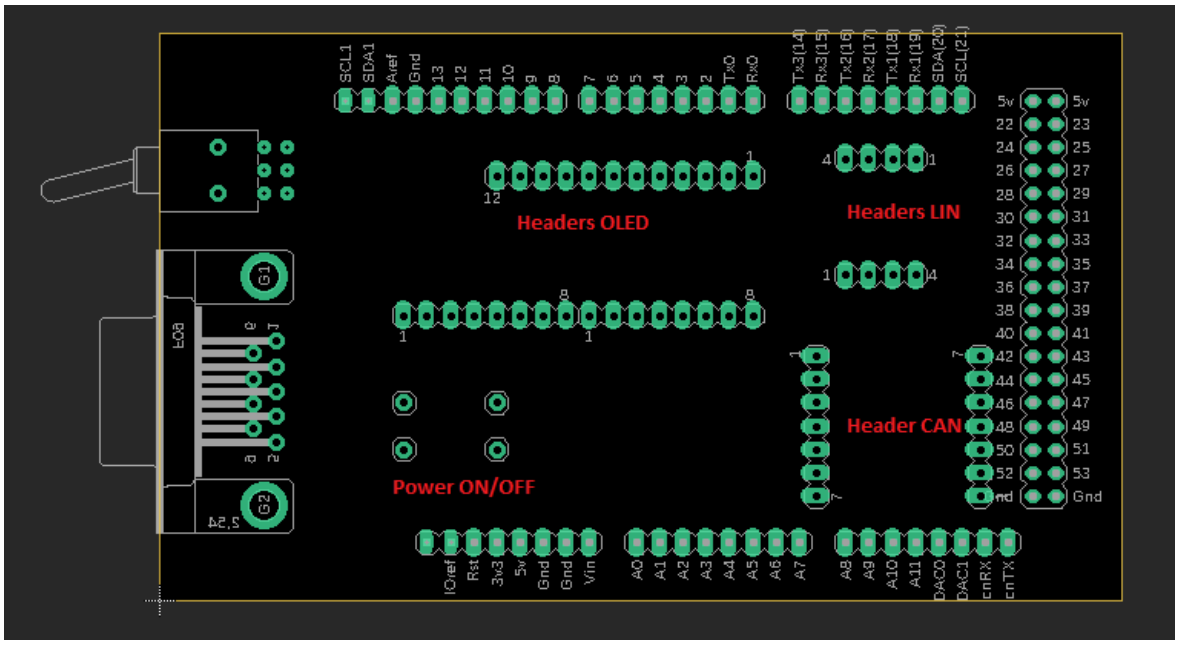


Fig. 18 - Componentes mecánicos. Fuente propia.

Los criterios seguidos para su ubicación han sido muy sencillos. En el caso de los headers CAN y LIN, están ubicados cerca de los pines CAN y LIN de Arduino, con motivo de facilitar su ruteado. La pantalla OLED sigue los mismos criterios.

En cuanto a los interruptores y conectores, debido a su gran tamaño, están ubicados en la arista vacía que tenemos en la PCB. De esta manera, quedan ambos en la misma sección del dispositivo.

4.4.3. Layout. Componentes y pistas.

Al ser un diseño de PCB a dos caras (bottom y top), se tienen la opción de utilizar componentes SMD o componentes convencionales de orificio pasante. En nuestro caso, se utilizan ambos por comodidad en el diseño.

Los componentes se pueden agrupar en tres bloques:

- Componentes potencia
- Componentes LIN
- Componentes CAN

A continuación, se detalla su colocación en cada bloque funcional.

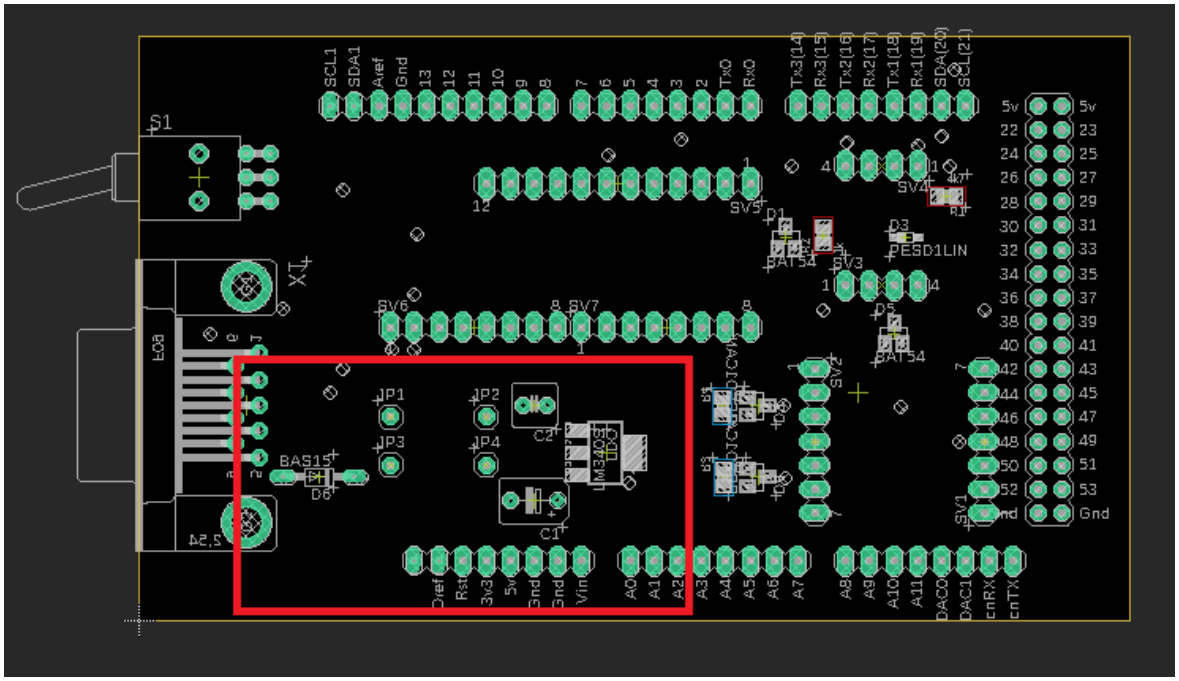


Fig. 19 - Bloque potencia. Fuente propia.

En la Fig.19, se puede observar el denominado bloque de potencia, que no es más que la entrada de alimentación de nuestro dispositivo, formada por un diodo, un interruptor y un regulador de voltaje. El regulador es SMD por cuestiones de tamaño principalmente, mientras que los demás componentes se han seleccionado de orificio pasante.

Dicho bloque se ha decidido ubicar en esa sección de la PCB por cercanía a los pines de Arduino a los que ha de ir conectado. Como se puede apreciar, se encuentra muy cercano a los pines de entrada Vin y GND. Todos los componentes están ubicados en la cara TOP.

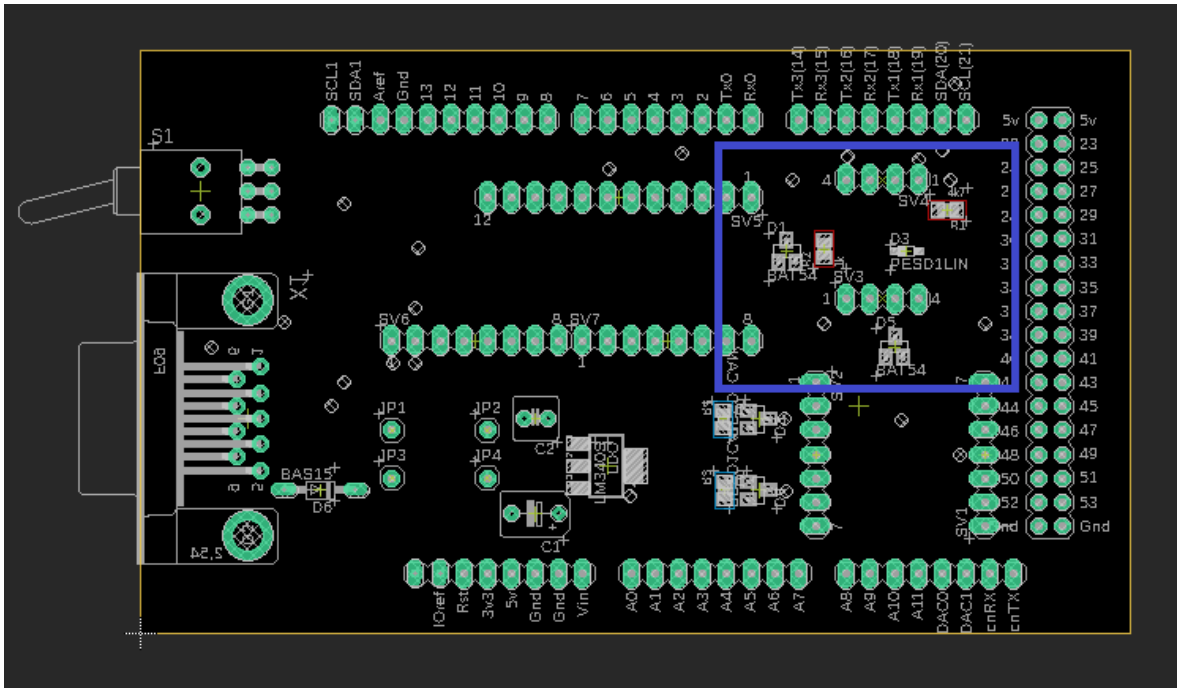


Fig. 20 - Bloque LIN. Fuente propia.

En la Fig. 20, se puede apreciar el bloque LIN. Estos componentes están ubicados cercanos al pin headers que hacen de zócalo para el transceptor LIN.

El bloque está formado por componentes SMD en su totalidad. Se trata de diodos y resistencias, ubicados en la cara TOP del diseño.

La ubicación del bloque es debido igual que en el caso anterior, a la cercanía con los pines Arduino elegidos para sus funciones.

Finalmente, el último de los bloques mencionados, lo podemos observar en la Fig. 21.

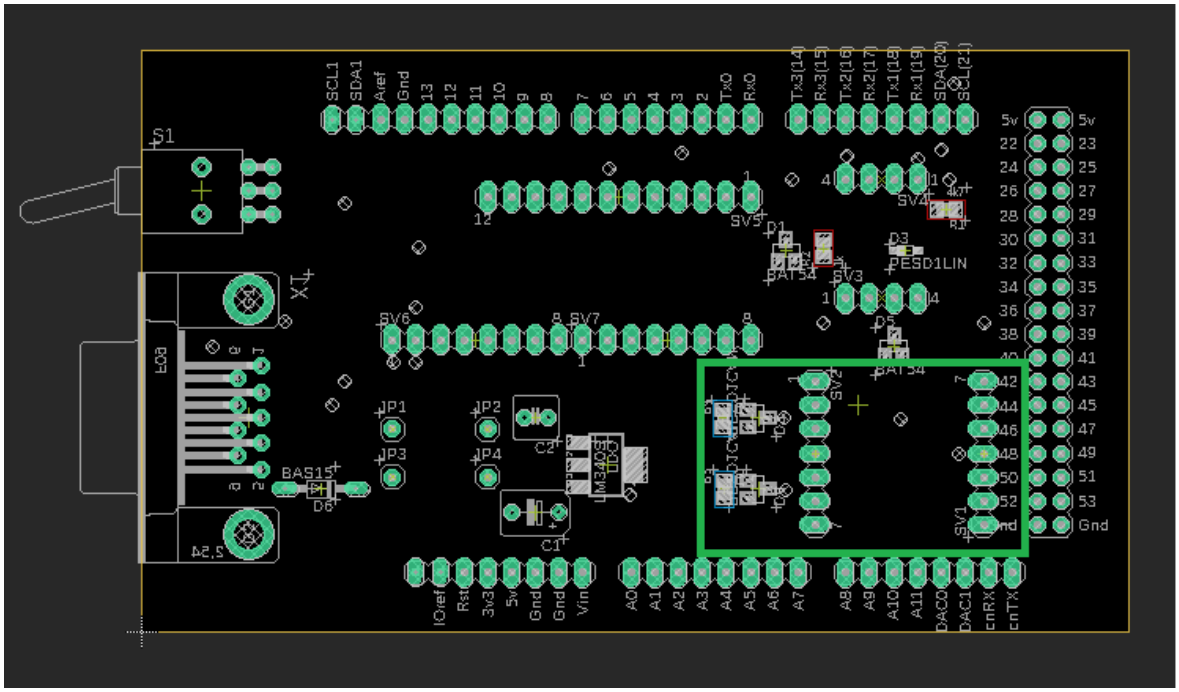


Fig. 21 - Bloque CAN. Fuente propia.

El bloque CAN, véase la Fig. 21, está formado por componentes SMD al igual que el anterior. En este caso son dos integrados que contienen una serie de diodos y dos resistencias. A diferencia de los bloques anteriores, los componentes están ubicados en la cara BOTTOM.

Una vez ubicados todos los componentes es el momento de rutear las pistas. Dado que disponemos de dos caras, se realizará de la siguiente manera.

En la Fig. 22 se puede observar la cara TOP ruteada.

Como criterio de diseño, en primer lugar, se rutea el bloque de potencia íntegramente en esta capa. Contiene también algunas conexiones correspondientes en su mayoría a la pantalla OLED, además de potencia para el bloque LIN.

En la Fig. 23 se muestra la capa BOTTOM, donde hay ruteadas la mayor parte de las conexiones con los interruptores y conectores, así como la mayor longitud de trazado de comunicaciones CAN y LIN.

Ambas capas son rellenadas con un plano de GND. Además, por cuestiones de trazado, se han tenido que utilizar vías para interconectar ambas capas y poder conseguir un trazado de la pista sin cruces ni intersecciones.

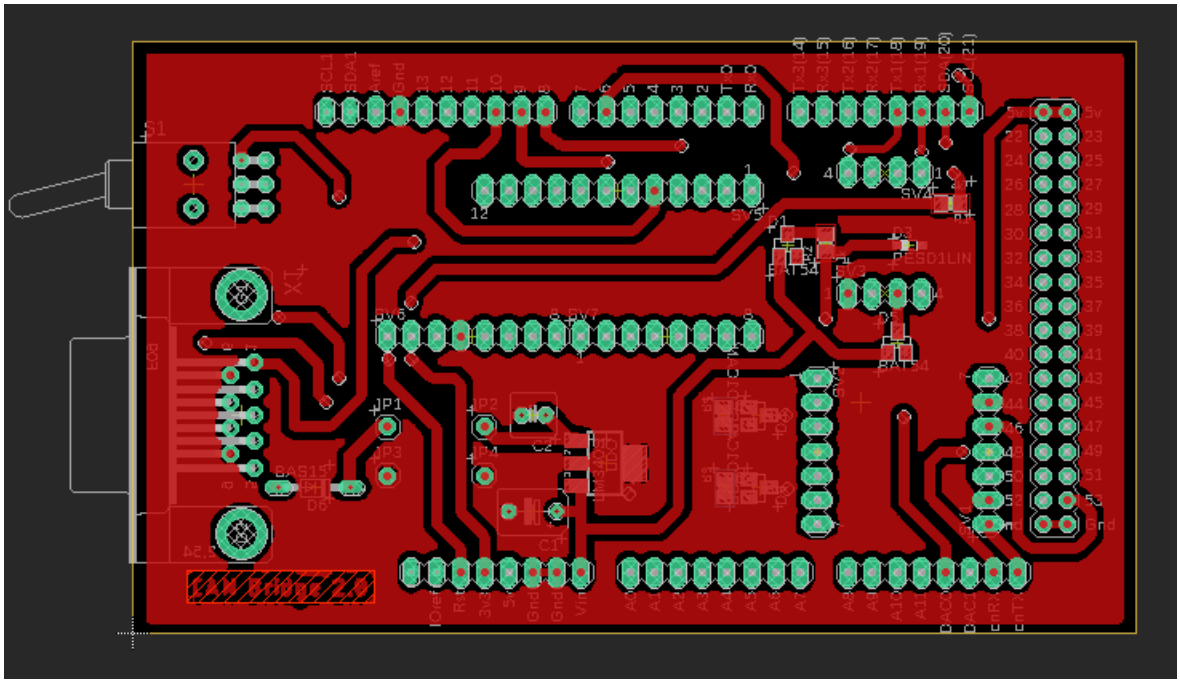


Fig. 22 - Cara TOP. Fuente propia.

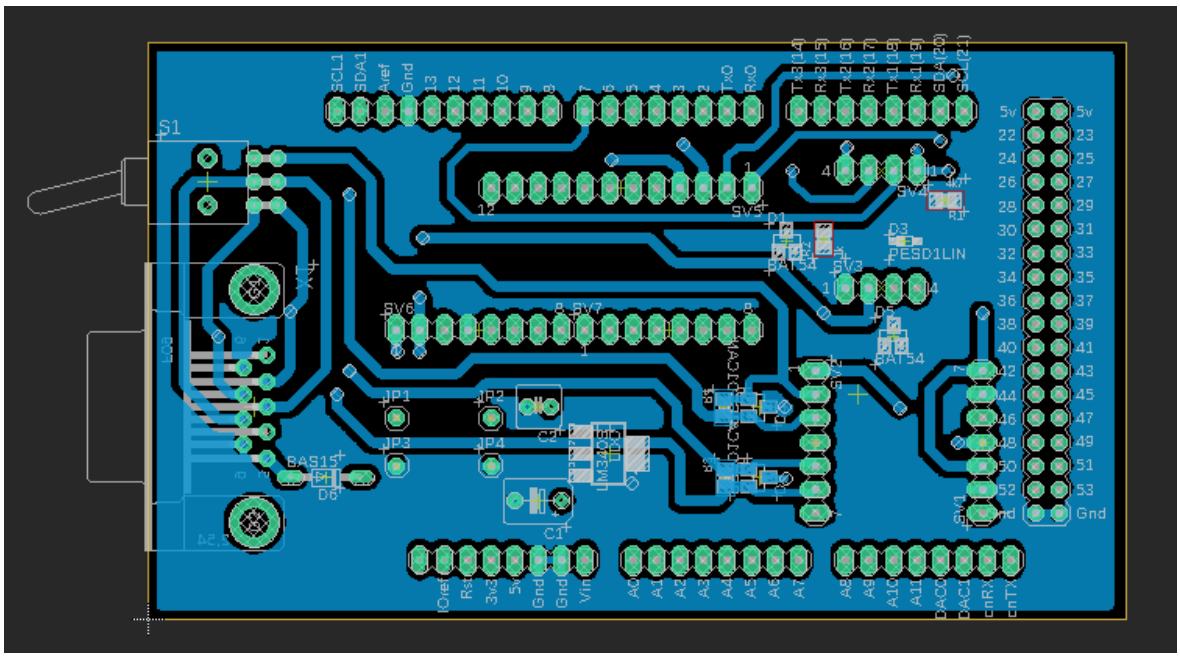


Fig. 23 - Capa BOTTOM. Fuente propia.

4.5. Interfaz de conexión

Esta interfaz la podríamos dividir en dos partes, ya que está formada por un conector y por un cable.

El conector se trata de un D-Sub de 9 vías hembra, que está montado en la PCB, como se ha mostrado anteriormente.

El pinout de este conector ha estado definido en función del diseño del layout, ya que, junto con el cable, forman un conjunto de conectores con pinout standard para los diferentes tipos de bus.

Seguidamente, se muestra en la Tabla 5 dicho pinout.

Pin	Señal
1	Voltaje batería (input)
2	Sin conexión
3	CAN Low saliente
4	CAN Low entrante
5	LIN Bus
6	GND
7	CAN High saliente
8	CAN High entrante
9	GND

Tabla 5 - Pinout D-Sub9. Fuente propia.

En cuanto al cable, su función es la de separar en varios conectores individuales cada canal de comunicaciones, por lo que quedaría definido de la siguiente manera. Falta pie de tabla…

Nombre del conector	**Placa**	**CAN1**	**CAN2**	**LIN**	**Alimentación**	
Tipo de conector	D-Sub 9 hembra	D-Sub 9 hembra	D-Sub 9 hembra	-	-	
Nombre señales	**pinout**	**pinout**	**pinout**	**pinout**	**pinout**	**COLOR**
Voltaje batería (input)	1				1	Blanco
Sin conexión	2					
CAN Low saliente	3		2			Verde
CAN Low entrante	4	2				Azul
LIN Bus	5			1		Marrón
GND	6				2	Negro
CAN High saliente	7		7			Amarillo
CAN High entrante	8	7				Rojo
GND	9			2		Gris

Tabla 6 - Definición cable interfaz. Fuente propia

5. Software

Una vez diseñado el hardware, las funcionalidades del dispositivo vendrán dadas mayoritariamente por el software que ejecutará dicho hardware. Ya que se dispone de bloques como una pantalla OLED, dos nodos CAN y un nodo LIN, en este proyecto se propone una aplicación que utilice todos estos bloques a modo de demostración. De cara a futuras aplicaciones más complejas, se podría adaptar el código o diseñar uno nuevo.

5.1. Diagrama de bloques

Como se ha descrito en el apartado 3.3, el dispositivo diseñado dispone de dos modos de funcionamiento en cuanto a comunicación CAN se refiere, radicando la diferencia principal en la utilización de uno o dos nodos CAN simultáneamente. De cara a la demostración del sistema, se propone utilizar un nodo CAN y el nodo LIN.

La idea es enviar a través de un PC un mensaje CAN por el canal 1, leerlo, procesarlo y contestar mediante LIN. En esta contestación incluiremos un par de cambios en los bytes de datos para mostrar la capacidad de procesamiento.

Haciendo referencia a la Fig. 324, se tiene un funcionamiento de la siguiente forma:

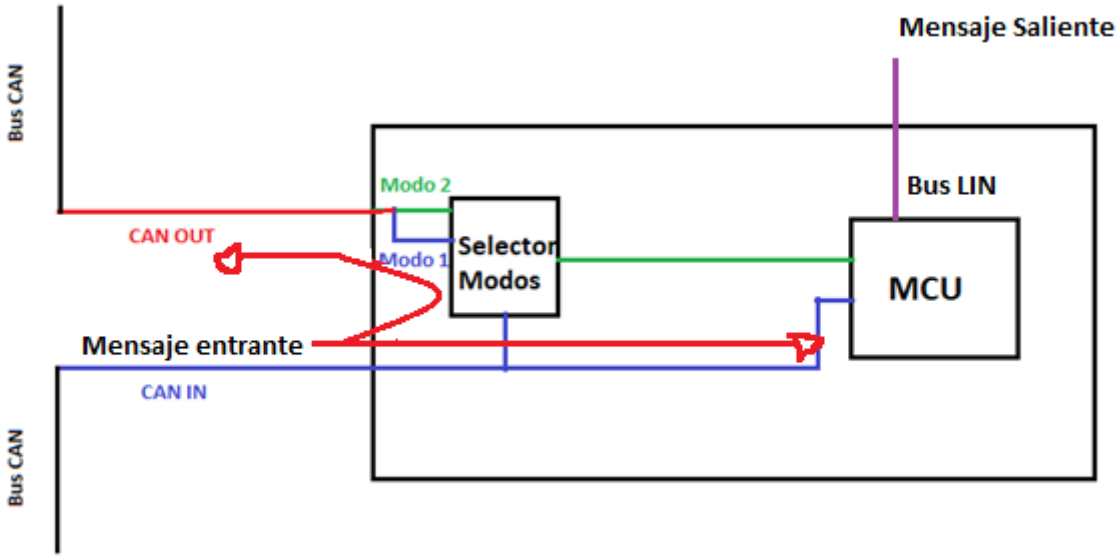


Fig. 24 - Funcionamiento aplicación. Fuente propia.

Al estar en modo 1, no hace falta leer el CAN2, ya que estaremos replicando exactamente el mensaje entrante por CAN1. En el caso que se quisiera ampliar la aplicación hacia una con funcionalidades completas, habilitaríamos mediante el selector de modos el modo 2, y podríamos enviar por CAN2.

A continuación, se muestra un diagrama de bloques del funcionamiento lógico de la aplicación a desarrollar.

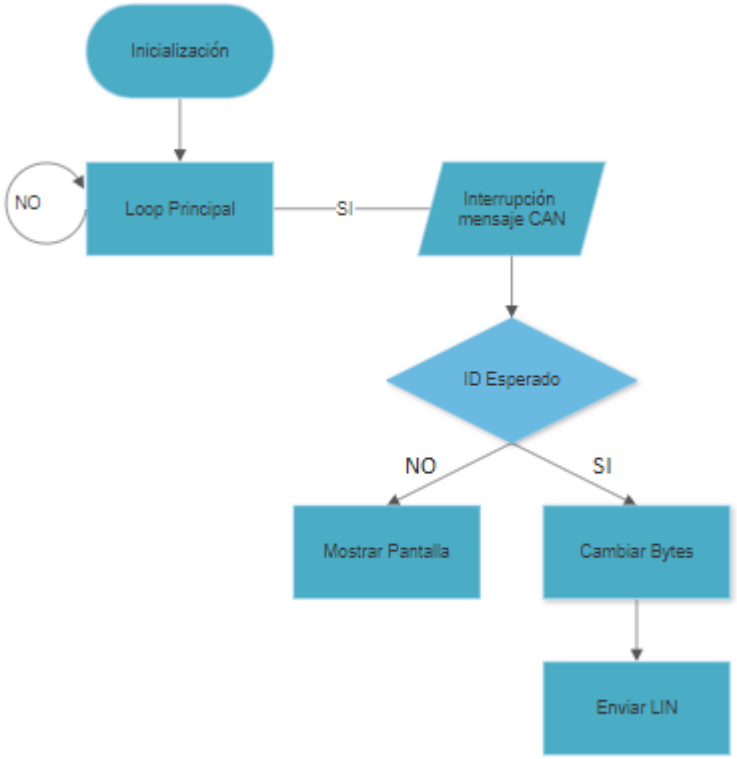


Fig. 25 - Diagrama de bloques. Fuente propia.

Como puede apreciarse en la Fig. 25, el código está compuesto por una inicialización y un bucle principal (loop). En el bucle principal, el programa está a la espera de que, mediante interrupción, entre un mensaje CAN.

Una vez recibido el mensaje entrante, se lee el identificador ID para ver si es el mensaje que nos interesa. En función de esta discriminación, se actuará mostrando un mensaje en pantalla o enviando el mensaje modificado por LIN.

5.2. Código de la aplicación

A partir del diagrama de bloques anterior se procede a la elaboración del código de la aplicación.

No se entra en excesivo detalle con las instrucciones, ya que se adjunta el código comentado en el anexo, pero sí se van a definir las instrucciones claves de cada bloque del diagrama.

5.2.1. Inicialización

La inicialización únicamente se ejecuta una vez al arrancar el programa y es la sección donde definimos todas las librerías a utilizar, así como las variables generales.

```
#include <lin_stack.h>        //LIN library
#include "variant.h"
#include <due_can.h>          //CAN library
#include <SPI.h>              //I2C library
#include <Wire.h>             //I2C library
#include <Adafruit_GFX.h>     //OLED library
#include <Adafruit_SSD1306.h> //OLED library

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels
#define OLED_RESET     -1// Reset pin # (or -1 if sharing Arduino reset
pin)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

//OLED buttons
#define A 10
#define B 9
#define C 8

//CAN Standby
#define STBY0 A8
#define STBY1 A9

lin_stack LIN1(1); // Creating LIN Stack objects, 1 - first channel

byte datos[8];
```

Como puede apreciarse, la primera parte se centra en incluir en el programa las librerías que se van a utilizar. Acto seguido, se definen los parámetros del display OLED para posteriormente, poder iniciarlo correctamente.

Nombrar los pines de Arduino con una denominación propia ayuda a la hora de escribir el código, tal y como se aprecia en las instrucciones de definición de botones y standby.

Por último, se realiza una llamada a una instrucción para inicializar el LIN en el canal 1, además de definir una variable donde se guardarán los datos de las tramas.

Después de este primer encabezamiento, entraría la función de setup, que es obligatoria en la plataforma Arduino., Esta función, a diferencia del bucle principal, solo se ejecuta una vez al inicializar el dispositivo.

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  // Initialize Display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);  // initialize with the I2C
addr 0x3C (for the 128x32)
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0,0);
  display.print("CAN BRIDGE 2.0");
  display.display();

  // Initialize CAN0, Set the proper baud rates here
  Can0.begin(CAN_BPS_500K);
  //standard
  Can0.setRXFilter(0, 0x40F, 0x7FF, false);
  Can0.setRXFilter(0, 0, false); //catch all mailbox - no mailbox ID
specified

    //now register all of the callback functions.
  Can0.setCallback(0, gotFrameMB0);
  Can0.setGeneralCallback(gotFrame);

  pinMode (7, OUTPUT);
  digitalWrite(7, HIGH);
  pinMode (6, OUTPUT);
  digitalWrite(6, HIGH);
  pinMode(STBY0, OUTPUT);
  pinMode(STBY1, OUTPUT);
  digitalWrite(STBY0,LOW);
  digitalWrite(STBY1,LOW);
}
```

Dentro de esta función setup, las primeras instrucciones hacen referencia a la inicialización del OLED, además de mostrar en pantalla un mensaje.

Seguidamente, aparecen las instrucciones para inicializar el CAN., Se definen, entre otros, la tasa de baudios, así como los buzones donde irán a parar los mensajes recibidos. Se ha definido un buzón específico para el identificador ID que queremos identificar y uno genérico donde irán a parar los demás mensajes con otros IDs.

Por último, se fuerzan al nivel lógico requerido los pines de standby del CAN, así como los pines del transceptor LIN que lo requieran.

5.2.2. Funciones de acción.

Una vez inicializado los dispositivos, se escriben las funciones que realizan una acción por sí solas. Estas funciones son 3 en nuestro caso: el bucle loop de Arduino, el cual no se puede omitir y las funciones llamadas por los buzones al recibir un mensaje CAN.

Empezando por las funciones CAN, se muestra como ejemplo la función que modifica la trama y envía por LIN.

```
void gotFrameMB0(CAN_FRAME *frame)
{
   byte package[8] ={frame->data.bytes[0], frame->data.bytes[1],frame-
>data.bytes[2],frame->data.bytes[3],
   frame->data.bytes[4],frame->data.bytes[5],frame->data.bytes[6],frame-
>data.bytes[7]};
   Serial.println("");
   Serial.println("");
   display.clearDisplay();
   display.setCursor(0,0);
   display.setTextSize(1);
   display.print("CAN DATA");
   display.setCursor(0,10);
   for(int i = 0; i < 8; i++)
  {
  Serial.print(package[i],HEX);
  Serial.print(" ");
  display.print(package[i],HEX);
  display.print(" ");
  }
   Serial.println ("Prueba");
   package[7]=0x00;
   display.print("SENDING LIN...");
   display.display();
   LIN1.write(0x40F, package, 8); // Write data to LIN
}
```

Como se puede apreciar en la función superior, el funcionamiento es simple. Esta función es llamada por interrupción cuando entra un mensaje CAN en el buzón correspondiente.

Primero de todo, se lee la trama de datos y se muestra impresa por el monitor serie para poder comprobar, en el caso de tener un PC conectado con nuestro dispositivo, la trama recibida.

Seguidamente, se modifica el ultimo byte de la trama de datos y se genera un mensaje LIN con el identificador marcado y el paquete recibido por CAN modificado. Durante todo el proceso se muestra por la pantalla OLED la trama de datos entrante y el envío por LIN.

La otra función de CAN es muy similar a ésta, pero más simple. El proceso de lectura de la trama de datos es el mismo, pero en este caso, no se envía nada por LIN ni modificamos nada, sino que lo mostramos por el puerto serie y por la OLED.

6. Test y validación

Las principales funciones del dispositivo han sido probadas y validadas previamente al desarrollo de la aplicación completa, por lo que se han desarrollado pequeños fragmentos de código para poder testear y validar estos bloques.

Los bloques a testear han sido aparte de la aplicación completa, la comunicación CAN y la comunicación LIN.

6.1. Test comunicación CAN

Nuestro dispositivo cuenta con dos nodos CAN, por lo que debemos verificar el correcto funcionamiento de nuestra electrónica en cuanto a comunicación CAN se refiere.

Para ello se va a realizar una prueba de envío y recepción de mensajes por este bus en ambos canales.

6.1.1. Código bajo test

Para realizar el test del dispositivo diseñado se debe escribir un código que reciba un mensaje CAN por uno de los canales y lo vuelva a enviar, modificando algún parámetro o no.

Se dispone del hardware diseñado, que consiste en el Arduino Due y el transceptor CAN dual MCP25612FD. Existe una librería de comunicación CAN especialmente preparada para el modelo Due denominada due_can **¡Error! No se encuentra el origen de la referencia.**, así que es la que se utiliza en este proyecto.

El primer paso de todos es incluir la librería en el programa, por lo que se hace uso de la siguiente instrucción:

```
#include <due_can.h>
```

Debemos tener en cuenta que, por la configuración de nuestro transceptor, el pin de standby ha de forzarse a nivel bajo, ya que en caso contrario nunca entraríamos en modo operación. Lo podemos comprobar en la tabla de la Fig. 27, extraída del mismo datasheet.

Mode	STBYx Pin	R_{XDX} Pin	
		Low	High
Normal	Low	Bus is dominant	Bus is recessive
Standby	High	Wake-up request is detected	No wake-up request detected

Fig. 26 - Modos de operación MCP25612FD. Fuente [7].

Mirando nuestro esquemático, comprobamos a qué pines de Arduino se han conectado estos pines del transceptor y se fuerzan a nivel bajo.

```
#define STBY0 A8
#define STBY1 A9

// Initialize SBTY CAN transceiver
  pinMode(STBY0, OUTPUT);
  pinMode(STBY1, OUTPUT);
  digitalWrite(STBY0,LOW);
  digitalWrite(STBY1,LOW);
```

Estas instrucciones son clave para el correcto funcionamiento del transceptor. Acto seguido se procede a inicializar el nodo CAN y prepararlo para la recepción y envío de mensajes.

En el caso de esta verificación de funcionamiento, la metodología a utilizar será por polling o consulta. De cara a la aplicación final y para un mejor funcionamiento se utilizarán interrupciones.

```
// Initialize CAN0 and CAN1, Set the proper baud rates here
  Can0.begin(CAN_BPS_500K);
  //Can1.begin(CAN_BPS_500K);
    //By default there are 7 mailboxes for each device that are RX boxes
  //This sets each mailbox to have an open filter that will accept
extended
  //or standard frames
  int filter;
//  standard
  for (int filter = 3; filter < 7; filter++) {
    Can0.setRXFilter(filter, 0, 0, false);
    //Can1.setRXFilter(filter, 0, 0, false);
```

Con estas instrucciones se inicializa el dispositivo a la velocidad seleccionada y se prepara el CAN0 para recibir y enviar mensajes.

Dentro del bucle principal, mediante las instrucciones siguientes se consulta si existe un mensaje entrante y se vuelve a enviar modificando dos bytes.

```
if (Can0.available() > 0) {
    Can0.read(incoming);
  printFrame(incoming);
  if (incoming.id == 0x50) {
       incoming.data.bytes[0] = 0x27;
       incoming.data.bytes[1] = 0x27;
       Can0.sendFrame(incoming);
      }
else{
  Can0.sendFrame(incoming);
  }
```

Este código se prueba también para el canal 2, substituyendo `Can0` por `Can1` en el programa.

6.1.2. Prueba de funcionamiento

Para comprobar el funcionamiento, disponemos de un convertidor CAN-USB para PC, mediante el cual se podrá disponer de un nodo CAN en nuestro ordenador y así poder enviar y recibir mensajes del bus.

Este dispositivo en concreto es el Kvaser Leaf Light HS de la compañía Kvaser [14]. Junto con el software gratuito proporcionado por el fabricante, Kvaser CANKing, se dispone de un nodo CAN en nuestro PC.

Sin entrar en muchos detalles técnicos, el Kvaser Leaf Light HS es un dispositivo USB que dota de una interfaz CAN completa al ordenador donde se conecte. Podemos encontrar más información en la página del fabricante [15].

Fig. 27 - Kvaser Leaf Light HS. Fuente [15].

Como se ha visto anteriormente, el código de test modifica los mensajes con ID 0x50, cambiando sus dos primeros bytes a 0x27. Mediante Kvaser, lo que se va a hacer es enviar a nuestro dispositivo un mensaje de CAN con este identificador en concreto y ver si nos lo devuelve con estos bytes cambiados.

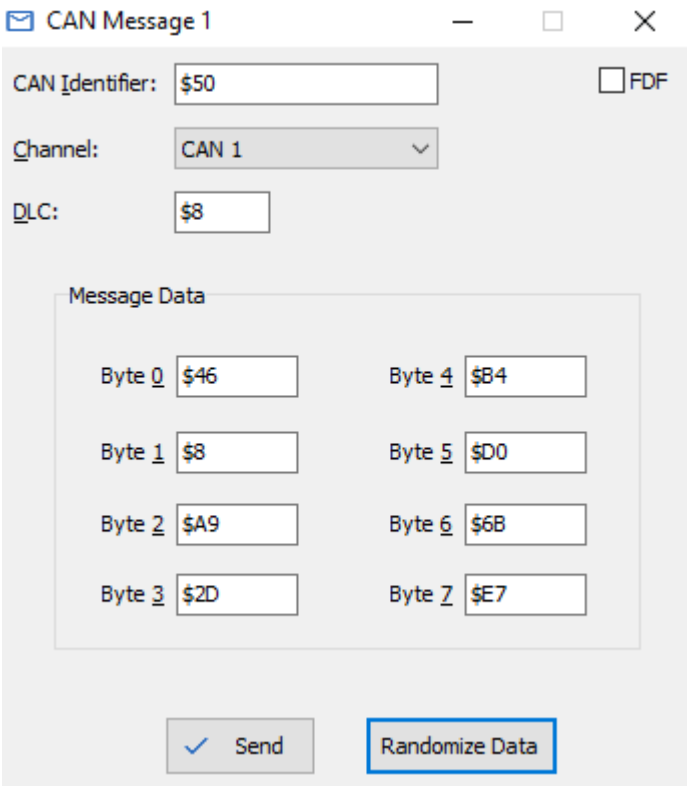


Fig. 28 - Mensaje CAN a enviar. Fuente propia.

En la 8, se puede ver el mensaje que vamos a enviar por el bus, debiendo recibir el mismo mensaje, pero con los siguientes cambios:

- **Byte 0**: 0x46 por 0x27
- **Byte 1:** 0x8 por 0x27

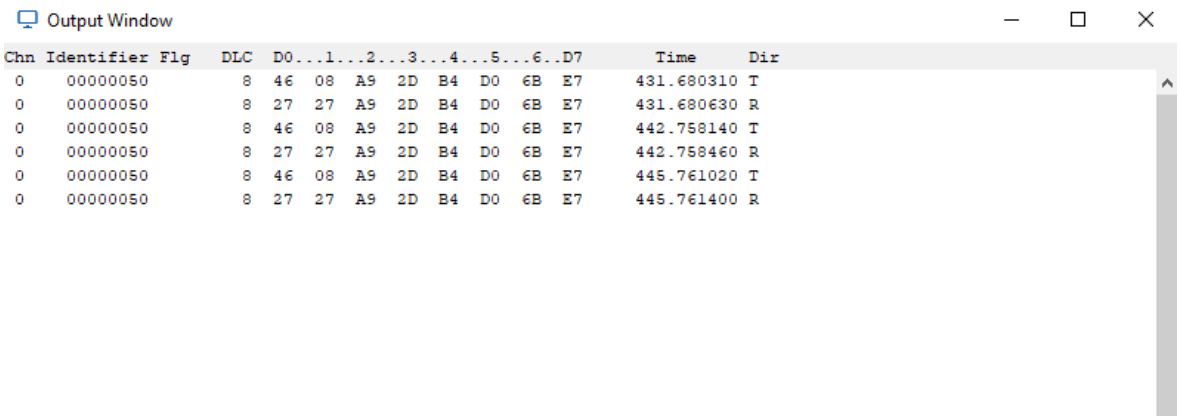


Fig. 29 - Ventana de registro Kvaser. Fuente propia.

La Fig. 29 corresponde con la ventana donde el software CANKing lleva el registro de los mensajes que se envían y se reciben por el bus. Como podemos apreciar, se han enviado tres mensajes con el identificador 0x50, mensajes con la T, y se han recibido los mismos con los bytes correspondientes cambiados, bytes 0 y 1, mensajes con la R.

Por lo tanto, para el primer canal CAN queda verificado su funcionamiento en envío y recepción.

Se ha repetido la prueba para el canal 2, cambiando en el código los parámetros mencionados anteriormente y se han obtenido los mismos resultados.

Con todos estos ensayos podemos dar por validada la comunicación CAN de nuestro dispositivo. En el anexo encontraremos el código completo que se ha comprobado.

6.2. Test comunicación LIN

Nuestro dispositivo constituye un nodo LIN completo, por lo que la manera más eficaz de comprobar su funcionamiento es el envío de una trama LIN para su verificación. Al no disponer de un analizador de LIN, se debe interpretar manualmente la trama a través de un osciloscopio.

6.2.1. Código bajo test

Para realizar el test de nuestro dispositivo, se debe programar el microcontrolador para enviar una trama periódicamente por LIN.

Como se ha visto anteriormente, se dispone de un Arduino Due y un transceptor MCP2004, el mismo Arduino dispone de una librería descargable para facilitar el uso de esta comunicación, por lo que se ha utilizado para este proyecto.

La librería en concreto se llama lin_stack [12].

A través de las instrucciones siguientes, se incluye dicha librería e se inicializa la comunicación LIN en el canal 1.

```
// Include LIN Stack library
#include <lin_stack.h>

lin_stack LIN1(1); // Creating LIN Stack objects, 1 - first channel
```

Se debe tener en cuenta que el transceptor LIN tiene dos pines que se deben manipular antes de empezar la comunicación. Estos pines son FAULT y CS, que están conectados a los pines 6 y 7 de Arduino, respectivamente.

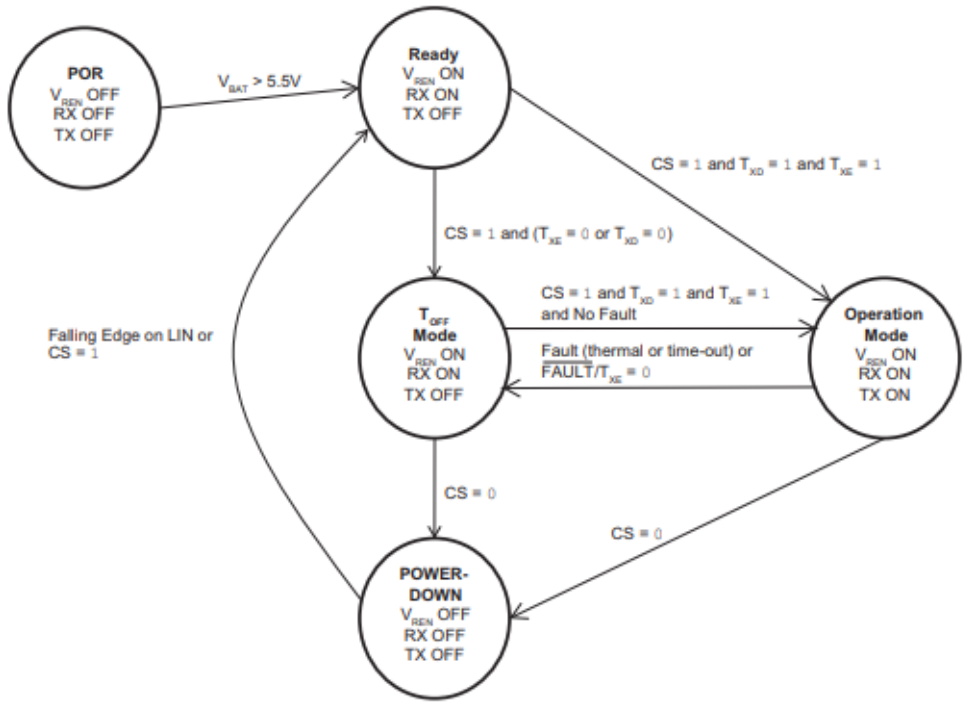


Fig. 30 - Diagrama de estados MCP2004. Fuente [8].

Como se puede apreciar en la Fig. 30, para llegar al modo de operación, CS y FAULT han de tener un valor de 1. Por lo tanto, en nuestro código:

```
pinMode (7, OUTPUT);
digitalWrite(7, HIGH);
pinMode (6, OUTPUT);
digitalWrite(6, HIGH);
```

Una vez puestos estos pines a nivel alto, se procede a enviar una trama. Para ello, se define un paquete de datos que se va a enviar y un identificador. Mediante la instrucción LIN1.write se envía la trama.

```
// Create Data Package
  byte package[5] = {31, 255, 0, 0, 3}; // byte package defined
  LIN1.write(0x15, package, 5); // Write data to LIN
```

El código complete se puede encontrar en el anexo correspondiente.

6.2.2. Análisis de la trama enviada

Una vez programado el Arduino Due con el código elaborado, se conecta un osciloscopio al conector asignado al bus LIN.

Mediante el osciloscopio se registran los datos de la trama y se grafican con Microsoft Excel para un mejor manejo e interpretación. La trama obtenida es la siguiente:

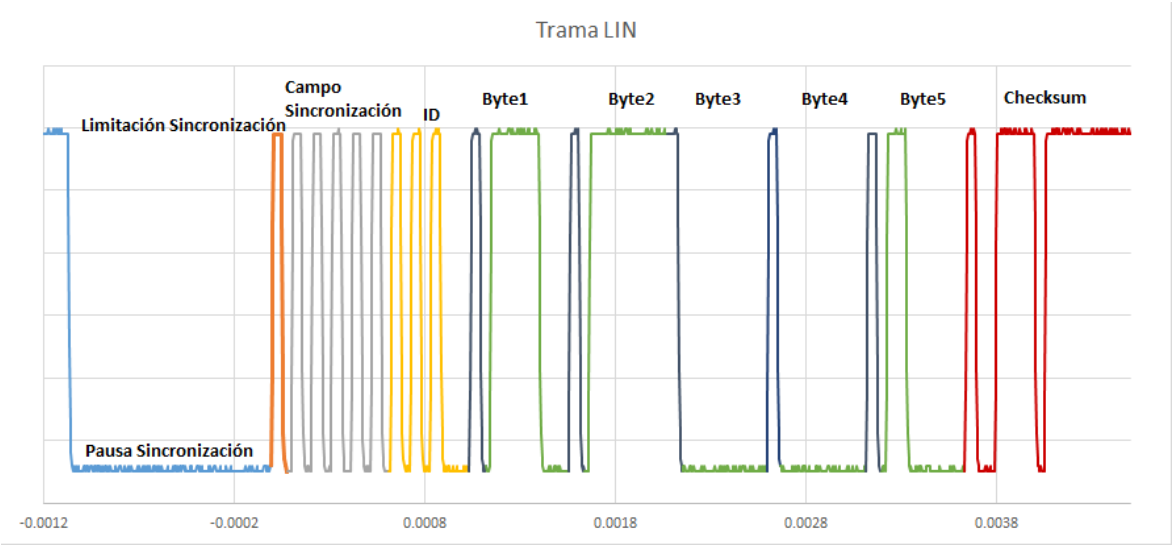


Fig. 31 - Trama LIN. Fuente propia.

En la Fig. 31 se muestra la trama obtenida. Se procede a analizar sus diferentes partes para comprobar si es correcta y corresponde con lo que hemos programado enviar.

Antes de esto, se debe entender como está formada una trama de LIN. Principalmente, está formada por dos partes, el encabezado o header y la respuesta o response.

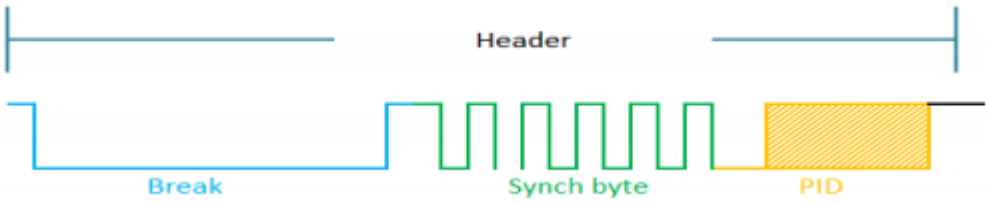


Fig. 32 - Header LIN. Fuente [11]

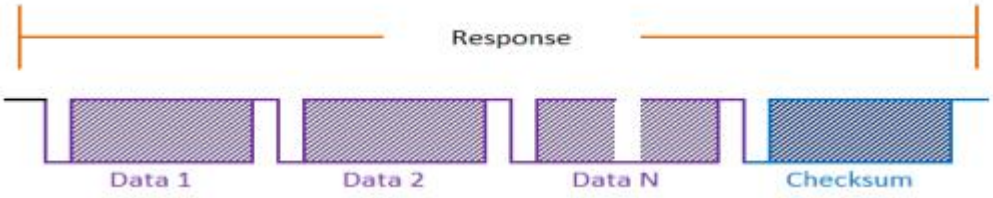


Fig. 33 - Response LIN. Fuente [11].

En las Fig. 32 y Fig. 33, se puede apreciar cómo se estructuran a su vez el encabezado y la respuesta. Teniendo en cuenta esto se puede proceder a analizar la trama registrada.

Encabezado

- **Pausa de sincronización o Break**: Compuesta por un mínimo de 13 bits a nivel dominante (0). En nuestro caso correspondería con la primera zona marcada en azul claro en la Fig. 31.
- **Limitación de sincronización**: La pausa de sincronización acaba con un bit a nivel recesivo (1). En nuestro caso de la Fig. 31, sería el inmediato bit marcado en naranja.
- **Campo de sincronización**: Compuesto por una serie de 10 bits alternando 0 y 1, (0101010101). Como vemos, correspondería con el segmento en gris de la Fig. 31.
- **ID**: Por último, el final del encabezado, correspondería con el identificador formado por 8 bits. En nuestro código enviamos un 0x15, que en binario correspondería con (00010101). En la Fig. 31, lo vemos marcado en amarillo y corresponde con el valor enviado.

Respuesta

Entre cada byte de dato que se envía, el protocolo marca que entre cada byte de envía un pulso a nivel recesivo (1) para indicar el inicio (start) del byte entrante y el final (stop) del byte anterior. Los datos son enviados, siguiendo el convenio de enviar primero el MSB.

Fig. 34 - Estructura datos LIN. Fuente [12]

- **Byte 1**: En nuestro código enviamos un 31 decimal, que en binario correspondería con (00011111). Como vemos en Fig. 31, tenemos 5 bits a 1 y 3 bits a 0. Byte correcto.
- **Byte 2**: En nuestro código enviamos un 255 decimal, que en binario correspondería con (11111111). Como vemos en Fig. 31, tenemos 8 bits a 1. Byte correcto.
- **Byte 3**: En nuestro código enviamos un 0 decimal, que en binario correspondería con (00000000). Como vemos en Fig. 31, tenemos 8 bits a 0. Byte correcto.
- **Byte 4**: En nuestro código enviamos un 0 decimal, que en binario correspondería con (00000000). Como vemos en Fig. 31, tenemos 8 bits a 0. Byte correcto.
- **Byte 5**: En nuestro código enviamos un 3 decimal, que en binario correspondería con (00000011). Como vemos en Fig. 31, tenemos 2 bits a 1 y 6 bits a 0. Byte correcto.
- **Checksum**: Por último, se envía el checksum para verificar la integridad del mensaje.

Habiendo analizado la trama registrada, se puede concluir que es correcta y por lo tanto la comunicación LIN de nuestro dispositivo queda verificada.

6.3. Validación aplicación completa

Para la validación de la aplicación completa, se necesita de más instrumentación que en los casos anteriores. Se van a utilizar dos canales del osciloscopio y el Kvaser HS Leaflight, como en la validación de la comunicación CAN.

El código bajo test es el código visto en el apartado de software de este mismo documento, 5.2.

El funcionamiento de la aplicación, como se ha descrito anteriormente, es la modificación de los datos de una trama CAN entrante, para su posterior envío mediante el protocolo LIN. Por lo tanto, para la validación, se va a registrar tanto el mensaje CAN entrante como el LIN saliente y se va a comparar su estructura de datos. Se deberá comprobar que uno de los bytes ha cambiado, tal y como manda el código.

6.3.1. Mensaje CAN

A través del software Kvaser CANKing, como en el caso anterior, se envía a nuestro dispositivo un mensaje CAN con las características siguientes.

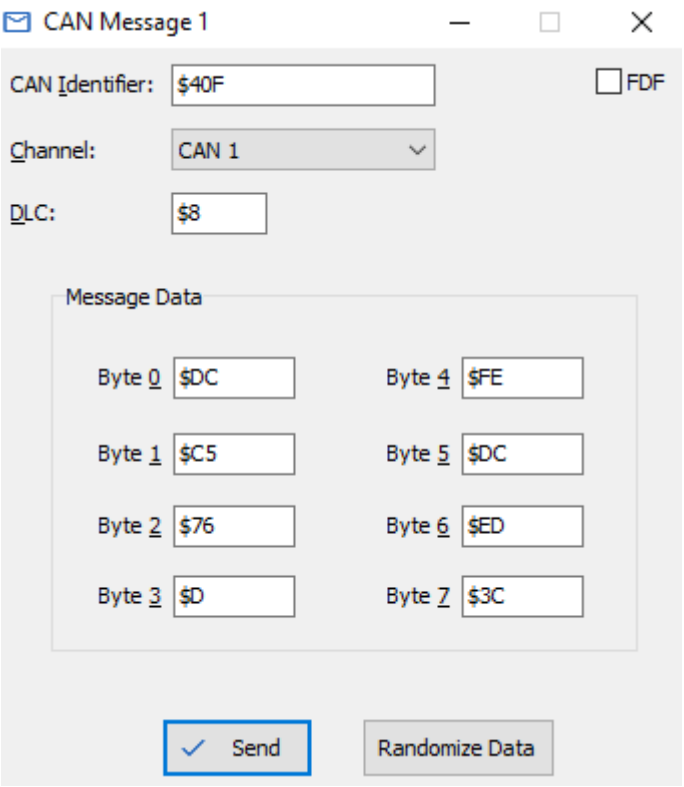


Fig. 35 - Mensaje CAN enviado. Fuente propia.

Como vemos, Fig. 35, se compone de una trama de 8 bytes que la podríamos descomponer de la manera siguiente.

- **ID:** 0x40F
- **Bytes:** 0xDC, 0xC5, 0x76, 0xD, 0xFE, 0xDC, 0xED, **0x3C**

Este mensaje lo recibe nuestro dispositivo y el código, como hemos visto en 5.2.2, modifica el ultimo byte de la trama por un 0x00.

En resumidas cuentas, nuestra trama LIN saliente debería presentar la estructura siguiente:

- **ID:** 0x40F
- **Bytes:** 0xDC, 0xC5, 0x76, 0xD, 0xFE, 0xDC, 0xED, **0x00**

6.3.2. Mensaje LIN

Para ello, mediante el osciloscopio, se captura el mensaje CAN enviado por Kvaser y el mensaje LIN enviado por nuestro dispositivo, de manera que podamos comparar su campo de datos.

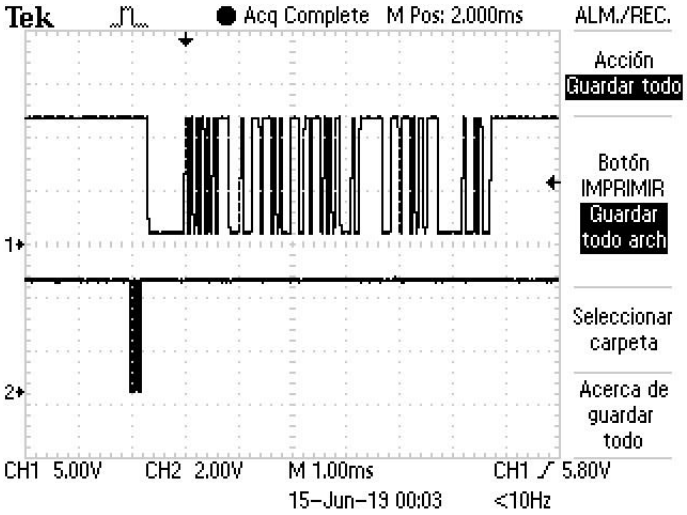


Fig. 36 - Trama CAN y LIN. Fuente propia.

En la Fig. 36, se muestra una captura del osciloscopio donde se muestra en la parte superior la trama LIN, y en la inferior la trama CAN. En este caso se puede apreciar cómo tras recibir una trama CAN, inmediatamente se envía la LIN.

A continuación, se muestra la trama LIN ampliada y procesada con Microsoft Excel para poder comprobar que, efectivamente, ha modificado los bytes que tocan.

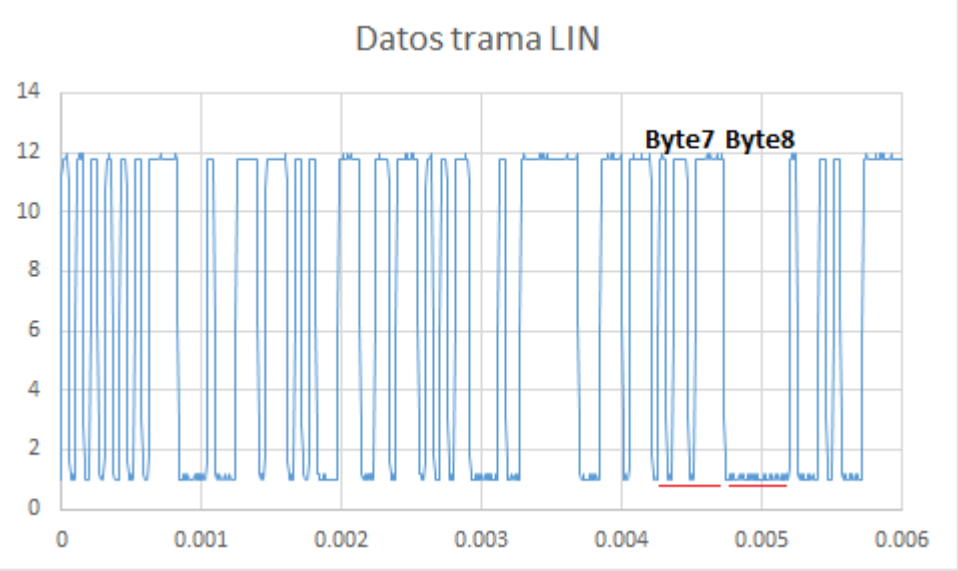


Fig. 37 - Datos trama LIN. Fuente propia.

Podemos apreciar como en la Fig. 37 hemos marcado los bytes 7 y 8 del campo de la respuesta.

El byte número 7 ha de seguir sin modificar, por lo que la conversión a binario del valor en hexadecimal 0xED (11101101), nos marcaría los bits a nivel alto o bajo, acabando por el MSB.

En cuanto al byte 8, como podemos apreciar, está totalmente a nivel dominante, por lo que marcaría 8 bits a 0, que justamente corresponden con el valor modificado por nuestro dispositivo.

El resto de bytes de datos han sido comprobados, así como el identificador. De cara a ilustrar la explicación nos hemos centrado en los dos últimos, que es donde podemos ver la intervención de la modificación de la respuesta programada.

En la siguiente página, se muestran las dos tramas de datos completas analizadas byte a byte, que si se comparan, se puede ver cómo en la Fig. 38, referente al CAN, tenemos una velocidad de 500 kbit/s así como dos bits de stuffing. Estos bits se dan en el protocolo CAN cuando hay 5 bits seguidos iguales, insertando un bit de stuffing con el valor opuesto a los 5 precedentes y se continua la comunicación.

En la Fig. 39 en cambio, tenemos los datos en la trama LIN. En esta ocasión no hay bits de stuffing y dentro del byte, el orden de los bits esta ordenado inversamente a la trama CAN. A diferencia de la trama CAN, en la comunicación LIN entre bytes, podemos ver un pulso de un bit de 1 a 0, que marcaría la separación entre cada byte.

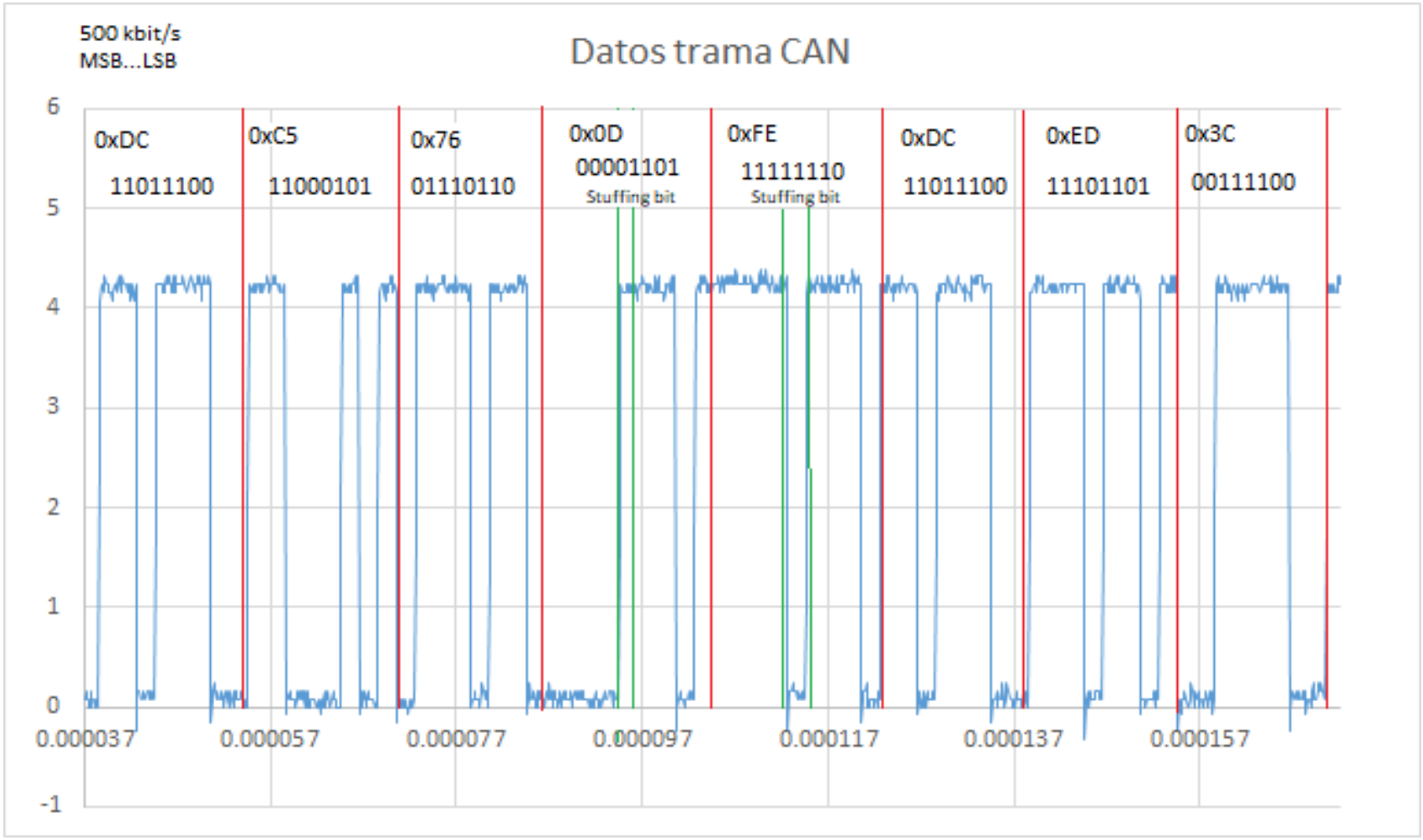


Fig. 38 - Datos CAN. Fuente propia.

ETSEIB

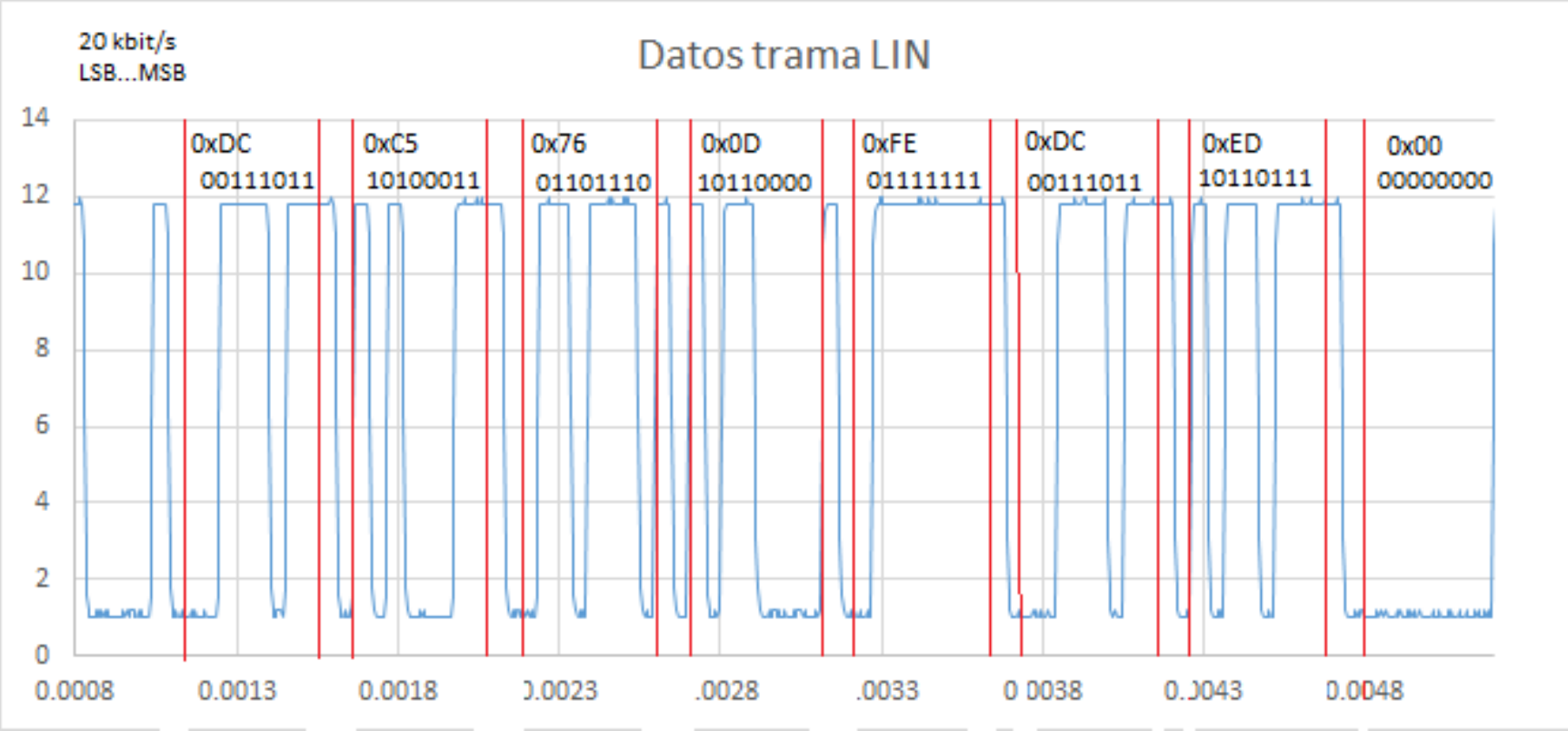


Fig. 39 - Datos LIN. Fuente propia.

7. Presupuesto

PRESUPUESTO CAN BRIDGE 2.0

Barcelona, 18/06/2019

GASTOS EN MATERIALES				
Descripción	**Tamaño lote**	**Coste/Lote**	**Unidades**	**Coste Total**
PCB	1	109,870 €	1	109,87 €
Arduino Due	1	36,390 €	1	36,39 €
D-Sub 9 vías Hembra	1	1,650 €	1	1,65 €
CAN Transceiver	10	0,900 €	10	9,00 €
LIN Transceiver	4	0,790 €	4	3,16 €
Diodo BAT54S	5	0,210 €	5	1,05 €
Diodo PESD1LIN	20	0,221 €	20	4,42 €
Diodo PESD1CAN	20	0,131 €	20	2,62 €
Diodo 1N4001	10	0,163 €	10	1,63 €
LM340S	5	1,578 €	5	7,89 €
Interruptor switch	1	4,100 €	1	4,10 €
Interruptor ON/OFF	1	4,060 €	1	4,06 €
D-Sub 9 vías Macho	1	1,450 €	1	1,45 €
OLED 128x32	1	14,300 €	1	14,30 €
Tira de pines	5	0,508 €	5	2,54 €
Tira de pines	5	0,848 €	5	4,24 €
Tira de pines	50	0,054 €	100	5,40 €
Adaptador IC	1	6,380 €	1	6,38 €
Adaptador IC	1	8,210 €	1	8,21 €
Resistencia 120 ohm	5	0,306 €	5	1,53 €
Resistencia 1k ohm	5	0,336 €	5	1,68 €
Condensador 0,1 uF	5	0,186 €	5	0,93 €
Condensador 0,22 uF	5	0,314 €	5	1,57 €
			Total	234,07 €

COSTES NO MATERIALES			
Concepto	**Coste/Hora**	**Horas**	**Coste Total**
Estudio previo	40,00 €	35	1.400,00 €
Desarrollo esquemático	40,00 €	45	1.800,00 €
Desarrollo layout	40,00 €	45	1.800,00 €
Compra materiales	40,00 €	5	200,00 €
Montaje PCB	40,00 €	15	600,00 €
Desarrollo software	40,00 €	25	1.000,00 €
Testeo y validación	40,00 €	55	2.200,00 €
Redacción memoria	40,00 €	75	3.000,00 €
		300	
		Total	12.000,00 €

GASTOS DE AMORTIZACIÓN				
Equipo	**Horas de uso**	**Coste**	**Años**	**Coste amortización**
Osciloscopio	55	650,00 €	5	0,82 €
Ordenador portatil	300	800,00 €	5	5,48 €
Kvaser Leaf Light HS	80	275,00 €	5	0,50 €
Soldador	15	50,00 €	5	0,02 €
			Total	6,82 €

Importe Neto	12.240,89 €
IVA (21%)	2.570,59 €
TOTAL FACTURA	**14.811,47 €**

8. Evaluación de impacto ambiental

Valorar el impacto ambiental que causa el proyecto es de gran importancia para poder considerar si necesita alguna modificación o por ende puede ser aplicado sin perjudicar el entorno.

El marco de este proyecto consiste en el desarrollo de un hardware para interactuar con un vehículo mediante las diversas comunicaciones que pueda presentar. Valorando los sistemas actuales, podemos ver que el dispositivo objeto de este proyecto es un dispositivo mucho más compacto y de funcionamiento autónomo.

Por lo tanto, al no necesitar de nada más que el mismo para funcionar, a diferencia de los actuales que pese a ser más potentes, necesitan de un PC, veríamos reducido el impacto ambiental, ya que estamos comparando un dispositivo compacto con dos o más y entre ellos un PC.

Agradecimientos

Quería agradecer la ayuda económica recibida por parte de mi familia para la fabricación y montaje del dispositivo objeto de este proyecto. Además, agradecer al profesor Manuel Moreno Eguílaz, tanto por su soporte académico durante la realización de este proyecto, así como por el préstamo de las herramientas necesarias para su test y validación.

Bibliografía

Referencias bibliográficas

[1] https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn1530/

Fecha de consulta: 05/03/2019

[2] https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn7640/

Fecha de consulta: 05/03/2019

[3] https://store.arduino.cc/due

Fecha de consulta: 05/03/2019

[4] https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Fecha de consulta: 05/03/2019

[5] https://cdn.instructables.com/FQJ/BZSV/H05NKJXO/FQJBZSVH05NKJXO.LARGE.jpg

Fecha de consulta: 29/03/2019

[6] https://upload.wikimedia.org/wikipedia/commons/thumb/4/4d/Canbus_levels.svg/220px-Canbus_levels.svg.png

Fecha de consulta: 03/04/2019

[7] http://ww1.microchip.com/downloads/en/DeviceDoc/20005409A.pdf

Fecha de consulta: 17/04/2019

[8] http://ww1.microchip.com/downloads/en/devicedoc/20002230g.pdf

Fecha de consulta: 17/04/2019

[9] https://docs-emea.rs-online.com/webdocs/1534/0900766b81534347.pdf

Fecha de consulta: 23/04/2019

[10] https://www.autodesk.com/products/eagle/overview

Fecha de consulta: 23/04/2019

[11] https://github.com/collin80/due_can

Fecha de consulta: 27/05/2019

[12] https://github.com/macchina/LIN/blob/master/src/lin_stack.cpp

Fecha de consulta: 29/05/2019

[13] https://rei.iteso.mx/bitstream/handle/11117/4093/HERRAMIENTA+DE+INSERCI%D3N+DE+ERRORES+EN+PROTOCOLO+LIN.pdf;jsessionid=4786800FC49555E075D27EDBBD2C4603?sequence=2

Fecha de consulta: 03/06/2019

[14] https://www.kvaser.com/about-can/can-standards/linbus/

Fecha de consulta: 04/06/2019

[15] https://www.kvaser.com/product/kvaser-leaf-light-hs/

Fecha de consulta: 07/06/2019